Paulina Jędrychowska, Rafał Kamiński, Marcin Kozub

Uprawnienia

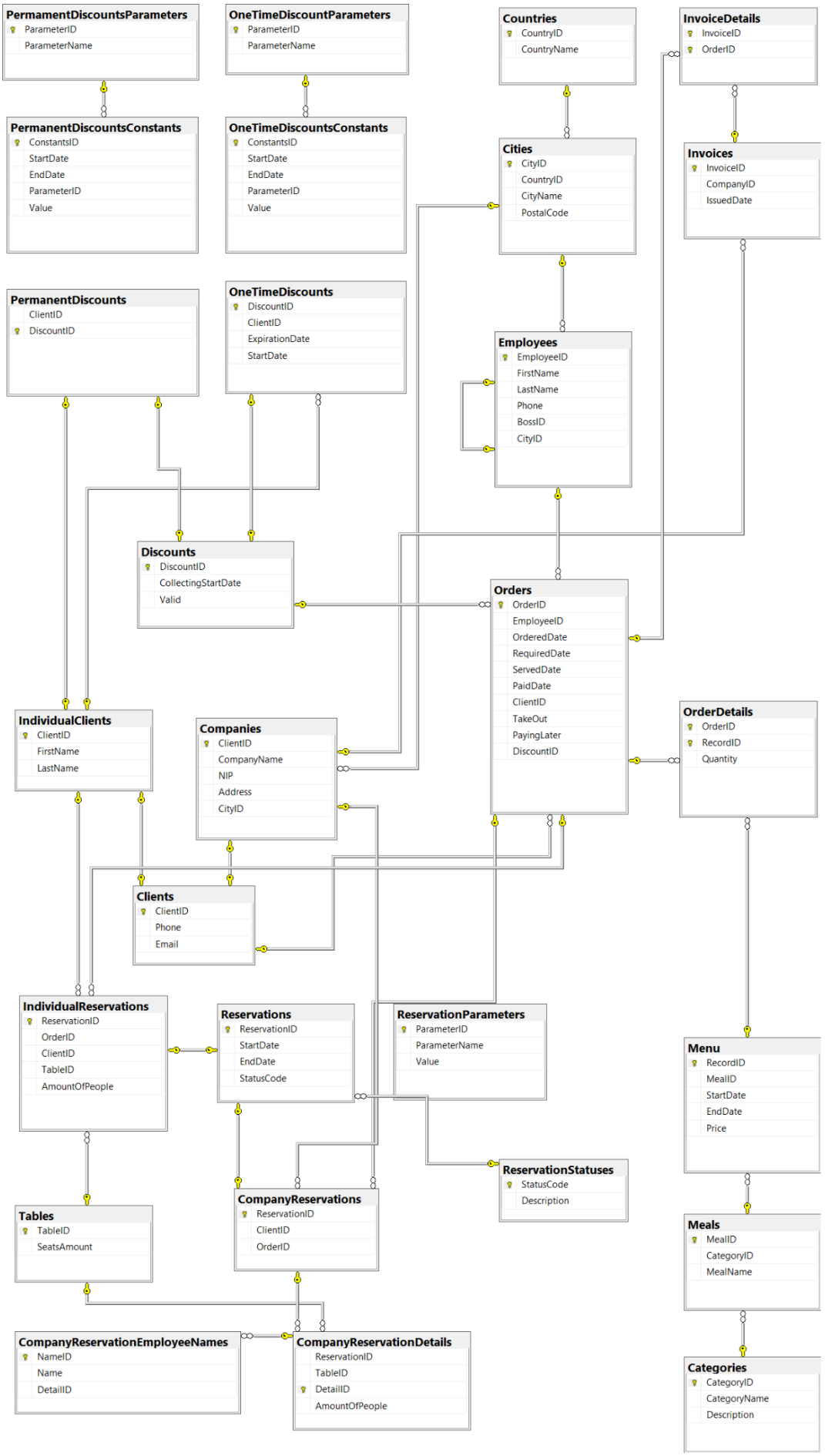
1. **System** - zarządzanie automatycznie wykonywanymi funkcjami
2. **Administrator Systemu** - całkowity dostęp do bazy danych,
3. **Właściciel restauracji** - dostęp do wszystkich danych i funkcji Pracowników, Klientów, Firm oraz do raportów.
4. **Menedżer** - dostęp do wszystkich danych i funkcji Właściciela restauracji z wyjątkiem zarządzania pracownikami,
5. **Pracownik** - dostęp do wszystkich danych o zamówieniach i funkcji obsługujących zamówienia
6. **Klient** - składanie zamówień, dokonywanie rezerwacji i możliwość generowania raportów miesięcznych i tygodniowych oraz dostęp do rabatów
7. **Firma** - dostęp do wszystkich funkcji Klienta oprócz dostępu do rabatów
8. **Pracownik firmy** - dostęp do wszystkich funkcji Klienta oprócz dostępu do rabatów oraz możliwość wzięcia faktury

Funkcje

1. dodawanie/usuwanie pracownika z bazy (Właściciel)
2. dodawanie nowego klienta do bazy (Pracownik, Klient)
3. dodawanie nowej firmy do bazy (Pracownik, Firma)
4. dodawanie nowego klienta z firmy zewnętrznej do bazy (Pracownik, Firma)
5. usuwanie klienta z firmy zewnętrznej z bazy (Pracownik, Firma)
6. dodawanie/anulowanie zamówienia do/z bazy (Pracownik, Klient, Firma, Pracownik firmy)
7. dodawanie/anulowanie zamówienia na wynos do/z bazy (Pracownik, Klient, Pracownik firmy)
8. tworzenie/usuwanie rezerwacji (Pracownik, Klient, Firma)
9. uaktualnienie daty rezerwacji (Pracownik, Klient, Firma)
10. dodanie nowego typu zniżki (Właściciel, Menedżer)
11. przypisanie zniżki klientowi/zamówieniu (System)
12. dodawanie/usuwanie stołu do bazy (Właściciel, Menedżer)
13. uaktualnienie informacji o stole w bazie (System)
14. tworzenie nowego menu (Właściciel, Menedżer)
15. dodanie/usuwanie/uaktualnienie pozycji do/w menu (Właściciel, Menedżer)
16. dodawanie/uaktualnienie posiłku do bazy/w bazie (Właściciel, Menedżer)
17. dodawanie/uaktualnienie kategorii posiłków do bazy/w bazie (Właściciel, Menedżer)
18. usuwanie niezapłaconego zamówienia/rezerwacji (Pracownik, Klient)
19. Wygenerowanie informacji o kliencie/firmie/pracowniku firmy (Pracownik, Klient, Firma, Pracownik firmy)
20. wygenerowanie listy aktualnych/wykorzystanych zniżek (Pracownik, Klient)
21. wygenerowanie listy aktualnych/wykonanych zamówień (Pracownik, Klient, Firma, Pracownik firmy)
22. wygenerowanie listy pracowników (Właściciel, Menedżer)
23. wygenerowanie szczegółów zamówienia (Pracownik, Klient, Firma, Pracownik firmy)
24. wygenerowanie listy rodzajów zniżek dostępnych dla klienta (Pracownik, Klient)

25. wygenerowanie listy wszystkich rodzajów zniżek (Pracownik)
26. wygenerowanie ilości miejsc przy danym stoliku (Pracownik, Klient, Firma)
27. wygenerowanie informacji o wolnych stolikach (Pracownik, Klient, Firma)
28. wygenerowanie informacji o wszystkich (zajętych) stolikach (Pracownik, Klient, Firma)
29. wygenerowanie informacji o wszystkich rezerwacjach (Pracownik)
30. wygenerowanie informacji o wszystkich rezerwacjach danego klienta (Pracownik, Klient, Firma)
31. wygenerowanie aktualnego menu (Pracownik, Klient, Firma, Pracownik firmy)
32. wygenerowanie poprzednich menu (Właściciel, Menedżer)
33. wygenerowanie listy zmienionych pozycji w menu (Właściciel, Menedżer)
34. wygenerowanie listy wszystkich posiłków/dań z owoców morza (Właściciel, Menedżer)
35. wygenerowanie informacji o posiłku (Pracownik, Klient, Firma, Pracownik firmy)
36. wygenerowanie listy kategorii posiłków (Właściciel, Menedżer)
37. wygenerowanie listy posiłków z danej kategorii posiłków (Właściciel, Menedżer)
38. wygenerowanie listy nieopłaconych zamówień (Pracownik)
39. wygenerowanie zysku uzyskanego z danego klienta (Właściciel)
40. wygenerowanie liczby rezerwacji z poprzedniego tygodnia/miesiąca (Właściciel, Klient, Firma)
41. wygenerowanie informacji o złożonych zamówieniach z poprzedniego tygodnia/miesiąca (Właściciel, Klient, Firma)
42. wygenerowanie daty zamówienia (Właściciel, Klient, Firma)
43. wygenerowanie oszczędzonej kwoty ze zniżek z poprzedniego tygodnia/miesiąca (Właściciel, Klient, Firma)
44. wygenerowanie faktury z poprzedniego tygodnia/miesiąca (Właściciel, Menedżer, Klient, Firma)
45. tworzenie backupów bazy danych (System)

Schemat bazy danych



Tabele

1. **Clients** - reprezentacja kluczu klienta w bazie danych
   Klucz główny: ClientID [typ int, wartosc nie może być nullem]
   adres e-mail: Email [typ varchar(50), wartość nie może być nullem]
   numer telefonu: Phone [typ varchar(50), wartość nie może być nullem]
   Warunki integralnościowe:
   - Phone musi się składać z samych cyfr i być unikalne:
     CHECK(UNIQUE([Phone])), CHECK ((isnumeric([Phone])=(1)))
   - Email musi być unikalne i zawierać znaki “@” i “.”:

   CHECK (([Email] like '%@%\.%'))

```
CREATE TABLE [dbo].[Clients](
	[ClientID] [int] NOT NULL,
	[Phone] [varchar](50) NOT NULL,
	[Email] [varchar](50) NOT NULL,
 CONSTRAINT [PK_Clients] PRIMARY KEY CLUSTERED
(
	[ClientID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY],
 CONSTRAINT [IX_Clients] UNIQUE NONCLUSTERED
(
	[Phone] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY],
 CONSTRAINT [IX_Clients_1] UNIQUE NONCLUSTERED
(
	[Email] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Clients]  WITH CHECK ADD  CONSTRAINT [ClientEmailConstraint]
CHECK  (([Email] like '%@%\.%'))
GO

ALTER TABLE [dbo].[Clients] CHECK CONSTRAINT [ClientEmailConstraint]
GO

ALTER TABLE [dbo].[Clients]  WITH CHECK ADD  CONSTRAINT [ClientPhoneConstraint]
CHECK  ((isnumeric([Phone])=(1)))
```

```
GO

ALTER TABLE [dbo].[Clients] CHECK CONSTRAINT [ClientPhoneConstraint]
GO
```

2. **IndividualClients** - reprezentacja indywidualnych klientów
   Klucz główny: ClientID [typ int, wartość nie może być nullem] (Klucz z tabeli Clients)
   imię: FirstName [typ varchar(50), wartość nie może być nullem]
   nazwisko: LastName [typ varchar(50), wartość nie może być nullem]
   Warunki integralnościowe:
   - FirstName i LastName nie może zawierać cyfr:

   CHECK ((NOT [FirstName] like '[0-9])), CHECK ((NOT [LastName] like '[0-9]))

```
CREATE TABLE [dbo].[IndividualClients](
	[ClientID] [int] NOT NULL,
	[FirstName] [varchar](50) NOT NULL,
	[LastName] [varchar](50) NOT NULL,
 CONSTRAINT [PK_IndividualClients] PRIMARY KEY CLUSTERED
(
	[ClientID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON
[PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[IndividualClients]  WITH CHECK ADD  CONSTRAINT
[Individual Client is Client] FOREIGN KEY([ClientID])
REFERENCES [dbo].[Clients] ([ClientID])
ON UPDATE CASCADE
ON DELETE CASCADE
GO

ALTER TABLE [dbo].[IndividualClients] CHECK CONSTRAINT [Individual Client is
Client]
GO

ALTER TABLE [dbo].[IndividualClients]  WITH CHECK ADD  CONSTRAINT
[CK_IndividualClients] CHECK  ((NOT [FirstName] like '[0-9]'))
GO

ALTER TABLE [dbo].[IndividualClients] CHECK CONSTRAINT
[CK_IndividualClients]
GO

ALTER TABLE [dbo].[IndividualClients]  WITH CHECK ADD  CONSTRAINT
```

```
[CK_IndividualClients_1] CHECK  ((NOT [LastName] like '[0-9]'))
GO

ALTER TABLE [dbo].[IndividualClients] CHECK CONSTRAINT
[CK_IndividualClients_1]
GO
```

3. **Companies** - reprezentacja firm zewnętrznych
   Klucz główny: ClientID [typ int, wartość nie może być nullem] (Klucz z tabeli Clients)
   Klucz obcy: CityID [typ int, wartość nie może być nullem] (Klucz z tabeli Cities)
   nazwa firmy: CompanyName [typ varchar(50), wartość nie może być nullem]
   numer identyfikacji podatkowej: NIP [typ varchar(50), wartość nie może być nullem]
   adres: Adress [typ varchar(50), wartość nie może być nullem]
   Warunki integralnościowe:
   - NIP musi być unikalne: *CHECK(UNIQUE([NIP]))*

```
CREATE TABLE [dbo].[Companies](
	[ClientID] [int] NOT NULL,
	[CompanyName] [varchar](50) NOT NULL,
	[NIP] [varchar](50) NOT NULL,
	[Address] [varchar](50) NOT NULL,
	[CityID] [int] NOT NULL,
 CONSTRAINT [PK_Companies] PRIMARY KEY CLUSTERED
(
	[ClientID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON
[PRIMARY],
 CONSTRAINT [IX_Companies] UNIQUE NONCLUSTERED
(
	[NIP] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON
[PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Companies]  WITH CHECK ADD  CONSTRAINT [Company
is Client] FOREIGN KEY([ClientID])
REFERENCES [dbo].[Clients] ([ClientID])
ON UPDATE CASCADE
ON DELETE CASCADE
GO
```

```
ALTER TABLE [dbo].[Companies] CHECK CONSTRAINT [Company is Client]
GO

ALTER TABLE [dbo].[Companies]  WITH CHECK ADD  CONSTRAINT [Company
is in City] FOREIGN KEY([CityID])
REFERENCES [dbo].[Cities] ([CityID])
ON UPDATE CASCADE
ON DELETE CASCADE
GO

ALTER TABLE [dbo].[Companies] CHECK CONSTRAINT [Company is in City]
GO
```

4. **PermanentDicounts** - reprezentacja zniżek na zawsze
   Klucz główny: DiscountID [typ int, wartość nie może być nullem] (Klucz z tabeli Discount)
   ważność zniżki: ClientID [typ int, wartość nie może być nullem] (Klucz z tabeli Clients)

```
CREATE TABLE [dbo].[PermanentDiscounts](
	[ClientID] [int] NOT NULL,
	[DiscountID] [int] NOT NULL,
 CONSTRAINT [PK_PermanentDiscounts_1] PRIMARY KEY CLUSTERED
(
	[DiscountID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON
[PRIMARY],
 CONSTRAINT [IX_PermanentDiscounts_1] UNIQUE NONCLUSTERED
(
	[ClientID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON
[PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[PermanentDiscounts]  WITH CHECK ADD  CONSTRAINT
[FK_PermanentDiscounts_Discounts] FOREIGN KEY([DiscountID])
REFERENCES [dbo].[Discounts] ([DiscountID])
GO
```

```
ALTER TABLE [dbo].[PermanentDiscounts] CHECK CONSTRAINT
[FK_PermanentDiscounts_Discounts]
GO

ALTER TABLE [dbo].[PermanentDiscounts]  WITH CHECK ADD  CONSTRAINT
[FK_PermanentDiscounts_IndividualClients] FOREIGN KEY([ClientID])
REFERENCES [dbo].[IndividualClients] ([ClientID])
GO

ALTER TABLE [dbo].[PermanentDiscounts] CHECK CONSTRAINT
[FK_PermanentDiscounts_IndividualClients]
GO
```

5. **PermanentDiscountsConstants**
   Klucz główny: ConstantsID [typ int, wartość nie może być nullem]
   Klucz obcy: ParameterID [typ int, wartość nie może być nullem] (Klucz z tabeli PermanentDiscountsParameters)
   wartość zniżki: Value [typ int, wartość nie może być nullem]
   data otrzymania zniżki: StartDate [typ datetime, wartość nie może być nullem]
   data utraty ważności zniżki: EndDate [typ datetime, wartość może być nullem]
   Warunki integralnościowe:
   - Value nie może być ujemne: *CHECK (([Value]>(0)))*
   - StartDate musi być datą wcześniejszą od EndDate: *CHECK(([StartDate]<[EndDate]))*

```
CREATE TABLE [dbo].[PermanentDiscountsConstants](
	[ConstantsID] [int] NOT NULL,
	[StartDate] [datetime] NOT NULL,
	[EndDate] [datetime] NULL,
	[ParameterID] [int] NOT NULL,
	[Value] [decimal](18, 2) NOT NULL,
 CONSTRAINT [PK_PermanentDiscountsConstants] PRIMARY KEY CLUSTERED
(
	[ConstantsID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[PermanentDiscountsConstants]  WITH CHECK ADD  CONSTRAINT
[FK_PermanentDiscountsConstants_PermamentDiscountsParemeters] FOREIGN
KEY([ParameterID])
REFERENCES [dbo].[PermamentDiscountsParameters] ([ParameterID])
```

```
GO

ALTER TABLE [dbo].[PermanentDiscountsConstants] CHECK CONSTRAINT
[FK_PermanentDiscountsConstants_PermamentDiscountsParemeters]
GO

ALTER TABLE [dbo].[PermanentDiscountsConstants]  WITH CHECK ADD  CONSTRAINT
[CK_PermanentDiscountsConstants] CHECK  (([EndDate] IS NULL OR
[StartDate]<[EndDate]))
GO

ALTER TABLE [dbo].[PermanentDiscountsConstants] CHECK CONSTRAINT
[CK_PermanentDiscountsConstants]
GO

ALTER TABLE [dbo].[PermanentDiscountsConstants]  WITH CHECK ADD  CONSTRAINT
[CK_PermanentDiscountsConstants_1] CHECK  (([Value]>(0)))
GO

ALTER TABLE [dbo].[PermanentDiscountsConstants] CHECK CONSTRAINT
[CK_PermanentDiscountsConstants_1]
GO
```

6. **PermanentDiscountsParameters**
   Klucz główny: ParameterID [typ int, wartość nie może być nullem]
   nazwa parametru: ParameterName [typ varchar(50), wartość nie może być nullem]
   Warunki integralnościowe:
   - ParameterName musi być unikalny:
     CHECK((UNIQUE([ParameterName])))

```
CREATE TABLE [dbo].[PermamentDiscountsParameters](
	[ParameterID] [int] NOT NULL,
	[ParameterName] [varchar](50) NOT NULL,
 CONSTRAINT [PK_PermamentDiscountsParemeters] PRIMARY KEY CLUSTERED
(
	[ParameterID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY],
 CONSTRAINT [IX_PermamentDiscountsParameters] UNIQUE NONCLUSTERED
(
	[ParameterName] ASC
```

```
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
```

7. **OneTimeDiscounts** - reprezentacja zniżek jednorazowych
   Klucz główny: DiscountID [typ int, wartość nie może być nullem] (Klucz obcy z tabeli Discounts)
   Klucz obcy ClientID [int not null] (Klucz z tabeli Clients)
   data utraty ważności zniżki: ExpirationDate [typ datetime, wartość może być nullem]
   data otrzymania zniżki: StartDate [typ datetime, wartość może być nullem]
   Warunki integralnościowe:
   - StartDate musi być datą wcześniejsza od ExpirationDate: *CHECK (([CollectingStartDate]<=[StartDate]))*

```
CREATE TABLE [dbo].[OneTimeDiscounts](
	[DiscountID] [int] NOT NULL,
	[ClientID] [int] NOT NULL,
	[ExpirationDate] [datetime] NULL,
	[StartDate] [datetime] NULL,
 CONSTRAINT [PK_Discounts] PRIMARY KEY CLUSTERED
(
	[DiscountID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON
[PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[OneTimeDiscounts]  WITH CHECK ADD  CONSTRAINT
[Client has discount] FOREIGN KEY([ClientID])
REFERENCES [dbo].[IndividualClients] ([ClientID])
GO

ALTER TABLE [dbo].[OneTimeDiscounts] CHECK CONSTRAINT [Client has
discount]
GO

ALTER TABLE [dbo].[OneTimeDiscounts]  WITH CHECK ADD  CONSTRAINT
[FK_OneTimeDiscounts_Discounts1] FOREIGN KEY([DiscountID])
REFERENCES [dbo].[Discounts] ([DiscountID])
GO
```

```
ALTER TABLE [dbo].[OneTimeDiscounts] CHECK CONSTRAINT
[FK_OneTimeDiscounts_Discounts1]
GO

ALTER TABLE [dbo].[OneTimeDiscounts]  WITH CHECK ADD  CONSTRAINT
[DateConstraint] CHECK  (([ExpirationDate]>[StartDate]))
GO

ALTER TABLE [dbo].[OneTimeDiscounts] CHECK CONSTRAINT [DateConstraint]
GO
```

8. **OneTimeDiscountsConstants**
   Klucz główny: ConstantsID [typ int, wartość nie może być nullem]
   Klucz obcy: ParameterID [typ int, wartość nie może być nullem] (Klucz z tabeli OneTimeDiscountsParameters)
   wartość zniżki: Value [typ bit, wartość nie może być nullem]
   data otrzymania zniżki: StartDate [typ datetime, wartość nie może być nullem]
   data utraty ważności zniżki: EndDate [typ datetime, wartość może być nullem]
   Warunki integralnościowe:
   - Value nie może być ujemne: *CHECK (([Value]>(0)))*
   - StartDate musi być datą wcześniejszą od EndDate: *CHECK(([EndDate]>[StartDate]))*

```
CREATE TABLE [dbo].[OneTimeDiscountsConstants](
	[ConstantsID] [int] NOT NULL,
	[StartDate] [datetime] NOT NULL,
	[EndDate] [datetime] NULL,
	[ParameterID] [int] NOT NULL,
	[Value] [decimal](18, 2) NOT NULL,
 CONSTRAINT [PK_OneTimeDiscountsConstants] PRIMARY KEY CLUSTERED
(
	[ConstantsID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[OneTimeDiscountsConstants]  WITH CHECK ADD  CONSTRAINT
[FK_OneTimeDiscountsConstants_OneTimeDiscountParameters] FOREIGN
KEY([ParameterID])
REFERENCES [dbo].[OneTimeDiscountParameters] ([ParameterID])
GO
```

```
ALTER TABLE [dbo].[OneTimeDiscountsConstants] CHECK CONSTRAINT
[FK_OneTimeDiscountsConstants_OneTimeDiscountParameters]
GO

ALTER TABLE [dbo].[OneTimeDiscountsConstants]  WITH CHECK ADD  CONSTRAINT
[CK_OneTimeDiscountsConstants] CHECK  (([EndDate] IS NULL OR
[EndDate]>[StartDate]))
GO

ALTER TABLE [dbo].[OneTimeDiscountsConstants] CHECK CONSTRAINT
[CK_OneTimeDiscountsConstants]
GO

ALTER TABLE [dbo].[OneTimeDiscountsConstants]  WITH CHECK ADD  CONSTRAINT
[CK_OneTimeDiscountsConstants_1] CHECK  (([Value]>(0)))
GO

ALTER TABLE [dbo].[OneTimeDiscountsConstants] CHECK CONSTRAINT
[CK_OneTimeDiscountsConstants_1]
GO
```

9. **OneTimeDiscountParameters**
   Klucz główny: ParameterID [typ int, wartość nie może być nullem]
   nazwa parametru: ParameterName [typ varchar(50), wartość nie może być nullem]

   Warunki integralnościowe:
   - ParameterName musi być unikalny:
     CHECK(UNIQUE([ParameterName]))

```
CREATE TABLE [dbo].[OneTimeDiscountParameters](
	[ParameterID] [int] NOT NULL,
	[ParameterName] [varchar](50) NOT NULL,
 CONSTRAINT [PK_OneTimeDiscountParameters] PRIMARY KEY CLUSTERED
(
	[ParameterID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY],
 CONSTRAINT [IX_OneTimeDiscountParameters] UNIQUE NONCLUSTERED
(
	[ParameterName] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
```

```
) ON [PRIMARY]
GO
```

10. Discounts

Klucz główny: *DiscountID*[typ int, wartość nie może być nullem]
nazwa parametru: *CollectingStartDate*[typ datetime, wartość nie może być nullem],
Valid [typ bit, wartość nie może być nullem]

```
CREATE TABLE [dbo].[Discounts](
	[DiscountID] [int] IDENTITY(1,1) NOT NULL,
	[CollectingStartDate] [datetime] NOT NULL,
	[Valid] [bit] NOT NULL,
 CONSTRAINT [PK_Discounts_1] PRIMARY KEY CLUSTERED
(
	[DiscountID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
```

11. Reservations - reprezentacja wszystkich rezerwacji

Klucz główny: ReservationID [typ int, wartość nie może być nullem]
Klucz obcy: StatusCode [typ int, wartość nie może być nullem] (Klucz z tabeli ReservationStatuses)
data początku rezerwacji: StartDate [typ datetime, wartosc nie może być nullem]
data końca rezerwacji: EndDate [typ datetime, wartość może być nullem]
Warunki integralnościowe:

- StartDate musi być datą wcześniejszą od EndDate: *CHECK (([EndDate] IS NULL OR [EndDate]>[StartDate]))*

```
CREATE TABLE [dbo].[Reservations](
	[ReservationID] [int] NOT NULL,
	[StartDate] [datetime] NOT NULL,
	[EndDate] [datetime] NULL,
	[StatusCode] [int] NOT NULL,
 CONSTRAINT [PK_Reservations] PRIMARY KEY CLUSTERED
(
	[ReservationID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON
[PRIMARY]
```

```
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Reservations]  WITH CHECK ADD  CONSTRAINT
[Reservation has status] FOREIGN KEY([StatusCode])
REFERENCES [dbo].[ReservationStatuses] ([StatusCode])
GO

ALTER TABLE [dbo].[Reservations] CHECK CONSTRAINT [Reservation has
status]
GO

ALTER TABLE [dbo].[Reservations]  WITH CHECK ADD  CONSTRAINT
[CK_Reservations] CHECK  (([EndDate] IS NULL OR [EndDate]>[StartDate]))
GO

ALTER TABLE [dbo].[Reservations] CHECK CONSTRAINT [CK_Reservations]
GO
```

12. **CompanyReservations** - reprezentacja rezerwacji od firm
    Klucz główny: ReservationID [typ int, wartość nie może być nullem] (Klucz z tabeli Reservations)
    Klucze obce: ClientID [typ int, wartość nie może być nullem] (Klucz z tabeli Clients)
    OrderID [typ int, wartość może być nullem] (Klucz z tabeli Orders)

```
CREATE TABLE [dbo].[CompanyReservations](
        [ReservationID] [int] NOT NULL,
        [ClientID] [int] NOT NULL,
        [OrderID] [int] NULL,
 CONSTRAINT [PK_CompanyReservations] PRIMARY KEY CLUSTERED
(
        [ReservationID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON
[PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[CompanyReservations]  WITH CHECK ADD  CONSTRAINT
[Company has reservation] FOREIGN KEY([ClientID])
REFERENCES [dbo].[Companies] ([ClientID])
GO

ALTER TABLE [dbo].[CompanyReservations] CHECK CONSTRAINT [Company
has reservation]
GO
```

```
ALTER TABLE [dbo].[CompanyReservations]  WITH CHECK ADD  CONSTRAINT
[Company Reservation has order] FOREIGN KEY([OrderID])
REFERENCES [dbo].[Orders] ([OrderID])
GO

ALTER TABLE [dbo].[CompanyReservations] CHECK CONSTRAINT [Company
Reservation has order]
GO

ALTER TABLE [dbo].[CompanyReservations]  WITH CHECK ADD  CONSTRAINT
[FK_CompanyReservations_Reservations] FOREIGN KEY([ReservationID])
REFERENCES [dbo].[Reservations] ([ReservationID])
GO

ALTER TABLE [dbo].[CompanyReservations] CHECK CONSTRAINT
[FK_CompanyReservations_Reservations]
GO
```

13. **CompanyReservationDetails** - reprezentacja szczegółów rezerwacji od firmy; przetrzymuje stoliki przypisane do danej rezerwacji
    Klucz główny: DetailID [typ int, wartość nie może być nullem]
    Klucze obce: ReservationID [typ int, wartość nie może być nullem] (Klucz z tabeli Reservations),
    TableID [typ int, wartość nie może być nullem] (Klucz z tabeli Tables)
    liczba ludzi przy rezerwacji: AmountOfPeople [typ int, wartość nie może być nullem]
    Warunki integralnościowe:
    - AmountOfPeople musi być dodatnie:
      CHECK(([AmountOfPeople]>(0)))

```
CREATE TABLE [dbo].[CompanyReservationDetails](
	[ReservationID] [int] NOT NULL,
	[TableID] [int] NULL,
	[DetailID] [int] NOT NULL,
	[AmountOfPeople] [int] NOT NULL,
 CONSTRAINT [PK_CompanyReservationDetails] PRIMARY KEY CLUSTERED
(
	[DetailID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON
[PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[CompanyReservationDetails]  WITH CHECK ADD
```

```
CONSTRAINT [Company reservation details] FOREIGN KEY([ReservationID])
REFERENCES [dbo].[CompanyReservations] ([ReservationID])
GO

ALTER TABLE [dbo].[CompanyReservationDetails] CHECK CONSTRAINT
[Company reservation details]
GO

ALTER TABLE [dbo].[CompanyReservationDetails]  WITH CHECK ADD
CONSTRAINT [Company reservation on tables] FOREIGN KEY([TableID])
REFERENCES [dbo].[Tables] ([TableID])
GO

ALTER TABLE [dbo].[CompanyReservationDetails] CHECK CONSTRAINT
[Company reservation on tables]
GO

ALTER TABLE [dbo].[CompanyReservationDetails]  WITH CHECK ADD
CONSTRAINT [CK_CompanyReservationDetails] CHECK
(([AmountOfPeople]>(0)))
GO

ALTER TABLE [dbo].[CompanyReservationDetails] CHECK CONSTRAINT
[CK_CompanyReservationDetails]
GO
```

14. CompanyReservationEmployeesNames

Klucz główny: NameID [typ int, wartość nie może być nullem]
Klucz obcy: DetailID [typ int, wartość nie może być nullem] (Klucz z tabeli CompanyReservationDetails)
imię i nazwisko pracownika firmy zewnętrznej: Name [typ varchar, wartość nie może być nullem]

```
CREATE TABLE [dbo].[CompanyReservationEmployeeNames](
        [NameID] [int] NOT NULL,
        [Name] [varchar](50) NOT NULL,
        [DetailID] [int] NOT NULL,
 CONSTRAINT [PK_CompanyReservationEmployeeNames] PRIMARY KEY CLUSTERED
(
        [NameID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
```

```
ALTER TABLE [dbo].[CompanyReservationEmployeeNames]  WITH CHECK ADD
CONSTRAINT [Company Employeees Details] FOREIGN KEY([DetailID])
REFERENCES [dbo].[CompanyReservationDetails] ([DetailID])
GO

ALTER TABLE [dbo].[CompanyReservationEmployeeNames] CHECK CONSTRAINT
[Company Employeees Details]
GO
```

15. **IndividualReservations** - reprezentacja rezerwacji od klientów indywidualnych
    Klucz główny: ReservationID [typ int, wartość nie może być nullem] (Klucz z tabeli Reservations)
    Klucze obce: OrderID [typ int, wartość nie może być nullem] (Klucz z tabeli Orders),
    ClientID [typ int, wartość nie może być nullem] (Klucz z tabeli Clients),
    TableID [typ int, wartość może być nullem] (Klucz z tabeli Tables)
    liczba ludzi przy rezerwacji: AmountOfPeople [typ int, wartość nie może być nullem]
    Warunki integralnościowe:
    - AmountOfPeople musi być większe lub równe 2:
      CHECK(([AmountOfPeople]>=(2)))

```
CREATE TABLE [dbo].[IndividualReservations](
	[ReservationID] [int] NOT NULL,
	[OrderID] [int] NOT NULL,
	[ClientID] [int] NOT NULL,
	[TableID] [int] NULL,
	[AmountOfPeople] [int] NOT NULL,
 CONSTRAINT [PK_IndividualReservations] PRIMARY KEY CLUSTERED
(
	[ReservationID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON
[PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[IndividualReservations]  WITH CHECK ADD  CONSTRAINT
[FK_IndividualReservations_Reservations] FOREIGN KEY([ReservationID])
REFERENCES [dbo].[Reservations] ([ReservationID])
GO

ALTER TABLE [dbo].[IndividualReservations] CHECK CONSTRAINT
[FK_IndividualReservations_Reservations]
GO
```

```
ALTER TABLE [dbo].[IndividualReservations]  WITH CHECK ADD  CONSTRAINT
[Individual client has reservation] FOREIGN KEY([ClientID])
REFERENCES [dbo].[IndividualClients] ([ClientID])
GO

ALTER TABLE [dbo].[IndividualReservations] CHECK CONSTRAINT [Individual
client has reservation]
GO

ALTER TABLE [dbo].[IndividualReservations]  WITH CHECK ADD  CONSTRAINT
[Individual reservation on table] FOREIGN KEY([TableID])
REFERENCES [dbo].[Tables] ([TableID])
GO

ALTER TABLE [dbo].[IndividualReservations] CHECK CONSTRAINT [Individual
reservation on table]
GO

ALTER TABLE [dbo].[IndividualReservations]  WITH CHECK ADD  CONSTRAINT
[Reservation has Order] FOREIGN KEY([OrderID])
REFERENCES [dbo].[Orders] ([OrderID])
GO

ALTER TABLE [dbo].[IndividualReservations] CHECK CONSTRAINT [Reservation
has Order]
GO

ALTER TABLE [dbo].[IndividualReservations]  WITH CHECK ADD  CONSTRAINT
[CK_IndividualReservations] CHECK  (([AmountOfPeople]>=(2)))
GO

ALTER TABLE [dbo].[IndividualReservations] CHECK CONSTRAINT
[CK_IndividualReservations]
GO
```

16. **ReservationStatuses** - reprezentacja statusu rezerwacji
    Klucz główny: StatusCode [typ int, wartość nie może być nullem]
    opis statusu: Description [typ varchar, wartość nie może być nullem]

```
CREATE TABLE [dbo].[ReservationStatuses](
        [StatusCode] [int] NOT NULL,
        [Description] [varchar](30) NOT NULL,
 CONSTRAINT [PK_ReservationStatuses] PRIMARY KEY CLUSTERED
(
        [StatusCode] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
```

```
ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON
[PRIMARY]
) ON [PRIMARY]
GO
```

17. **Orders** - reprezentacja zamówień
    Klucz główny: OrderID [typ int, wartość nie może być nullem]
    Klucz obcy: EmployeeID [typ int, wartość nie może być nullem] (Klucz z tabeli Employees) ,
    ClientID [typ int, wartość nie może być nullem] (Klucz z tabeli Clients)
    data złożenia zamówienia: OrderedDate [typ datetime, wartość nie może być nullem]
    oczekiwany czas odebrania zamówienia: RequiredDate [typ datetime, wartość nie może być nullem]
    data wydania zamówienia: ServedDate [typ datetime, wartość może być nullem]
    data płatności za zamówienie: PaidDate [typ datetime, wartość może być nullem]
    wartość zniżki na zamówienie: DiscountValue [typ decimal, wartość może być nullem]
    informacja czy zamowienie jest na wynos: TakeOut [typ bit, wartość nie może być nullem]
    informacja czy zamowienie jest oplacone: PayingLater [typ bit, wartość nie może być nullem]
    Warunki integralnościowe:
    - DiscountValue musi być z przedziału [0,1]: *CHECK(([DiscountValue]>=(0) AND [DiscountValue]<=(1)))*
    - OrderDate musi być datą wcześniejszą od reszty dat: RequiredDate, ServedDate i PaidDate: *CHECK (([OrderedDate]<[PaidDate]))*

```
CREATE TABLE [dbo].[Orders](
        [OrderID] [int] NOT NULL,
        [EmployeeID] [int] NOT NULL,
        [OrderedDate] [datetime] NOT NULL,
        [RequiredDate] [datetime] NOT NULL,
        [ServedDate] [datetime] NULL,
        [PaidDate] [datetime] NULL,
        [DiscountValue] [decimal](2, 2) NULL,
        [ClientID] [int] NULL,
        [TakeOut] [bit] NOT NULL,
        [PayingLater] [bit] NOT NULL,
 CONSTRAINT [PK_Orders] PRIMARY KEY CLUSTERED
(
        [OrderID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON
[PRIMARY]
) ON [PRIMARY]
```

```
GO

ALTER TABLE [dbo].[Orders]  WITH CHECK ADD  CONSTRAINT [Client makes
order] FOREIGN KEY([ClientID])
REFERENCES [dbo].[Clients] ([ClientID])
GO

ALTER TABLE [dbo].[Orders] CHECK CONSTRAINT [Client makes order]
GO

ALTER TABLE [dbo].[Orders]  WITH CHECK ADD  CONSTRAINT [Order was
serviced by Employee] FOREIGN KEY([EmployeeID])
REFERENCES [dbo].[Employees] ([EmployeeID])
ON UPDATE CASCADE
GO

ALTER TABLE [dbo].[Orders] CHECK CONSTRAINT [Order was serviced by
Employee]
GO

ALTER TABLE [dbo].[Orders]  WITH CHECK ADD  CONSTRAINT [CK_Orders]
CHECK  (([DiscountValue]>=(0) AND [DiscountValue]<=(1)))
GO

ALTER TABLE [dbo].[Orders] CHECK CONSTRAINT [CK_Orders]
GO

ALTER TABLE [dbo].[Orders]  WITH CHECK ADD  CONSTRAINT
[PaidDateConstraint] CHECK  (([PaidDate] IS NULL AND
[OrderedDate]<[PaidDate]))
GO

ALTER TABLE [dbo].[Orders] CHECK CONSTRAINT [PaidDateConstraint]
GO

ALTER TABLE [dbo].[Orders]  WITH CHECK ADD  CONSTRAINT
[RequiredDateConstraint] CHECK  (([OrderedDate]<[RequiredDate]))
GO

ALTER TABLE [dbo].[Orders] CHECK CONSTRAINT [RequiredDateConstraint]
GO

ALTER TABLE [dbo].[Orders]  WITH CHECK ADD  CONSTRAINT
[ServedDateConstraint] CHECK  (([ServedDate] IS NULL OR
[OrderedDate]<[ServedDate]))
GO

ALTER TABLE [dbo].[Orders] CHECK CONSTRAINT [ServedDateConstraint]
GO
```

18. **OrderDetails** - reprezentacja szczegółów zamówienia
    Klucz główny: (OrderID, RecordID) [oba typu int, wartości nie mogą być nullem]
    (Klucz OrderID z tabeli Orders, Klucz RecordID z tabeli Menu)
    cena zamówionej pozycji: Price [typ money, wartość nie może być nullem]
    ilość zamówionego pozycji: Quantity [typ int, wartość nie może być nullem]
    Warunki integralnościowe:
    - Price nie może być ujemne: *CHECK(([Price]>(0)))*
    - Quantity musi być dodatnie: *CHECK(([Quantity]>(0)))*

```
CREATE TABLE [dbo].[OrderDetails](
	[OrderID] [int] NOT NULL,
	[RecordID] [int] NOT NULL,
	[Price] [money] NOT NULL,
	[Quantity] [int] NOT NULL,
 CONSTRAINT [PK_OrderDetails] PRIMARY KEY CLUSTERED
(
	[OrderID] ASC,
	[RecordID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON
[PRIMARY],
 CONSTRAINT [IX_OrderDetails] UNIQUE NONCLUSTERED
(
	[OrderID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON
[PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[OrderDetails]  WITH CHECK ADD  CONSTRAINT [Order
details for Order] FOREIGN KEY([OrderID])
REFERENCES [dbo].[Orders] ([OrderID])
ON UPDATE CASCADE
ON DELETE CASCADE
GO

ALTER TABLE [dbo].[OrderDetails] CHECK CONSTRAINT [Order details for
Order]
GO

ALTER TABLE [dbo].[OrderDetails]  WITH CHECK ADD  CONSTRAINT [Order on
record] FOREIGN KEY([RecordID])
REFERENCES [dbo].[Menu] ([RecordID])
GO
```

```
ALTER TABLE [dbo].[OrderDetails] CHECK CONSTRAINT [Order on record]
GO

ALTER TABLE [dbo].[OrderDetails]  WITH CHECK ADD  CONSTRAINT
[OrderPriceConstraint] CHECK  (([Price]>(0)))
GO

ALTER TABLE [dbo].[OrderDetails] CHECK CONSTRAINT [OrderPriceConstraint]
GO

ALTER TABLE [dbo].[OrderDetails]  WITH CHECK ADD  CONSTRAINT
[OrderQuantityConstraint] CHECK  (([Quantity]>(0)))
GO

ALTER TABLE [dbo].[OrderDetails] CHECK CONSTRAINT
[OrderQuantityConstraint]
GO
```

19. **Tables** - reprezentacja stołów
    Klucz główny: TableID [typ int, wartość nie może być nullem]
    ilość dostępnych miejsc przy stole: SeatsAmount [typ int, wartość nie może być nullem]
    Warunki integralnościowe:
    - SeatsAmount musi być dodatnie: *CHECK(([SeatsAmount]>(0)))*

```
CREATE TABLE [dbo].[Tables](
        [TableID] [int] NOT NULL,
        [SeatsAmount] [int] NOT NULL,
 CONSTRAINT [PK_Tables] PRIMARY KEY CLUSTERED
(
        [TableID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON
[PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Tables]  WITH CHECK ADD  CONSTRAINT
[SeatsAmountConstraint] CHECK  (([SeatsAmount]>(0)))
GO

ALTER TABLE [dbo].[Tables] CHECK CONSTRAINT [SeatsAmountConstraint]
GO
```

20. **Meals** - słownik posiłków
    Klucz główny: MealID [typ int, wartość nie może być nullem]
    Klucz obcy: CategoryID [typ int, wartość nie może być nullem] (Klucz z tabeli Categories)
    nazwa posiłku: MealName [typ varchar, wartość nie może być nullem]
    Warunki integralnościowe:
    - MealName unikalne: *CHECK(UNIQUE(MealName))*

```
CREATE TABLE [dbo].[Meals](
	[MealID] [int] NOT NULL,
	[CategoryID] [int] NOT NULL,
	[MealName] [varchar](50) NOT NULL,
 CONSTRAINT [PK_Meals] PRIMARY KEY CLUSTERED
(
	[MealID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON
[PRIMARY],
 CONSTRAINT [IX_Meals] UNIQUE NONCLUSTERED
(
	[MealName] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON
[PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Meals]  WITH CHECK ADD  CONSTRAINT [Meal has
Category] FOREIGN KEY([CategoryID])
REFERENCES [dbo].[Categories] ([CategoryID])
GO

ALTER TABLE [dbo].[Meals] CHECK CONSTRAINT [Meal has Category]
GO
```

21. **Menu** - reprezentacja listy dostępnych posiłków w danym okresie
    Klucz główny: RecordID [typ int, wartość nie może być nullem]
    Klucz obcy: MealID [typ int, wartość nie może być nullem] (Klucz z tabeli Meals)
    data wstawienia posiłku do menu: StartDate [typ datetime, nie może być nullem]
    data usunięcia posiłku z menu: EndDate [typ datetime, nie może być nullem]
    cena za posiłek: Price [typ money, nie może być nullem]
    Warunki integralnościowe:
    - StartDate musi być datą wcześniejszą od EndDate: *CHECK([StartDate]<[EndDate])*
    - Price nie może być ujemne: *CHECK([Price]>=(0))*

```
CREATE TABLE [dbo].[Menu](
	[RecordID] [int] NOT NULL,
	[MealID] [int] NOT NULL,
	[StartDate] [datetime] NOT NULL,
	[EndDate] [datetime] NULL,
	[Price] [money] NOT NULL,
 CONSTRAINT [PK_MenuDetails] PRIMARY KEY CLUSTERED
(
	[RecordID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON
[PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Menu]  WITH CHECK ADD  CONSTRAINT [Menu position
contains Meal] FOREIGN KEY([MealID])
REFERENCES [dbo].[Meals] ([MealID])
GO

ALTER TABLE [dbo].[Menu] CHECK CONSTRAINT [Menu position contains Meal]
GO

ALTER TABLE [dbo].[Menu]  WITH CHECK ADD  CONSTRAINT
[MenuDateConstraint] CHECK  (([StartDate]<[EndDate]))
GO

ALTER TABLE [dbo].[Menu] CHECK CONSTRAINT [MenuDateConstraint]
GO

ALTER TABLE [dbo].[Menu]  WITH CHECK ADD  CONSTRAINT [Price cannot be
less than 0] CHECK  (([Price]>=(0)))
GO

ALTER TABLE [dbo].[Menu] CHECK CONSTRAINT [Price cannot be less than 0]
GO
```

22. **Categories** - reprezentacja kategorii posiłków
    Klucz główny: CategoryID [typ int, wartość nie może być nullem]
    nazwa kategorii: CategoryName [typ varchar(50), nie może być nullem]
    opis kategorii: Description [typ varchar(250), nie może być nullem]
    Warunki integralnościowe:
    - CategoryName jest unikalne: *CHECK(UNIQUE([CategoryName]))*

```
CREATE TABLE [dbo].[Categories](
```

```
	[CategoryID] [int] NOT NULL,
	[CategoryName] [varchar](50) NOT NULL,
	[Description] [varchar](250) NOT NULL,
 CONSTRAINT [PK_Categories_1] PRIMARY KEY CLUSTERED
(
	[CategoryID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON
[PRIMARY],
 CONSTRAINT [Category name is unique] UNIQUE NONCLUSTERED
(
	[CategoryName] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON
[PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Categories] ADD  CONSTRAINT
[DF_Categories_Description]  DEFAULT ('No description') FOR [Description]
GO
```

23. **Invoices** - reprezentacja faktur
    Klucz główny: ClientID [typ int, wartość nie może być nullem]
    Klucz obcy: CompanyID [typ int, wartość nie może być nullem] (Klucz z tabeli Companies)
    data wystawienia faktury: IssuedDate [typ datetime, nie może być nullem]

```
CREATE TABLE [dbo].[Invoices](
	[InvoiceID] [int] NOT NULL,
	[CompanyID] [int] NOT NULL,
	[IssuedDate] [datetime] NOT NULL,
 CONSTRAINT [PK_Invoices] PRIMARY KEY CLUSTERED
(
	[InvoiceID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON
[PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Invoices]  WITH CHECK ADD  CONSTRAINT [Invoice issued
for company] FOREIGN KEY([CompanyID])
REFERENCES [dbo].[Companies] ([ClientID])
GO
```

```
ALTER TABLE [dbo].[Invoices] CHECK CONSTRAINT [Invoice issued for
company]
GO
```

24. **InvoiceDetails** - reprezentacja szczegół faktury
    Klucz główny: (InvoiceID, OrderID) [oba typ int, nie mogą być nullem] (Klucze z tabeli Invoices i Orders)

```
CREATE TABLE [dbo].[InvoiceDetails](
        [InvoiceID] [int] NOT NULL,
        [OrderID] [int] NOT NULL,
 CONSTRAINT [PK_InvoiceDetails] PRIMARY KEY CLUSTERED
(
        [InvoiceID] ASC,
        [OrderID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON
[PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[InvoiceDetails]  WITH CHECK ADD  CONSTRAINT [Invoice
details] FOREIGN KEY([InvoiceID])
REFERENCES [dbo].[Invoices] ([InvoiceID])
GO

ALTER TABLE [dbo].[InvoiceDetails] CHECK CONSTRAINT [Invoice details]
GO

ALTER TABLE [dbo].[InvoiceDetails]  WITH CHECK ADD  CONSTRAINT [Invoice
issued for orders] FOREIGN KEY([OrderID])
REFERENCES [dbo].[Orders] ([OrderID])
GO

ALTER TABLE [dbo].[InvoiceDetails] CHECK CONSTRAINT [Invoice issued for
orders]
GO
```

25. **Employees** - reprezentacja pracowników restauracji
    Klucz główny: EmployeeID [typ int, nie może być nullem]
    Klucze obce: BossID, CityID [oba typu int, nie mogą być nullem] (klucz BossID jest z tabeli Employees, a klucz CityID z tabeli Cities)
    imię: FirstName [typ varchar(50), nie może być nullem]
    nazwisko: LastName [typ varchar(50) nie może być nullem]
    numer telefonu: Phone [typ varchar(50) nie może być nullem]

Warunki integralnościowe:

- Phone musi się składać z samych cyfr i być unikalne:
  CHECK(UNIQUE([Phone])), CHECK ((isnumeric([Phone])=(1)))
- FirstName i LastName nie może zawierać cyfr:

CHECK ((NOT [FirstName] like '[0-9]')), CHECK ((NOT [LastName] like '[0-9]'))

```
CREATE TABLE [dbo].[Employees](
	[EmployeeID] [int] NOT NULL,
	[FirstName] [varchar](50) NOT NULL,
	[LastName] [varchar](50) NOT NULL,
	[Phone] [varchar](50) NOT NULL,
	[BossID] [int] NOT NULL,
	[CityID] [int] NOT NULL,
 CONSTRAINT [PK_Employees] PRIMARY KEY CLUSTERED
(
	[EmployeeID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON
[PRIMARY],
 CONSTRAINT [IX_Employees] UNIQUE NONCLUSTERED
(
	[Phone] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON
[PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Employees]  WITH CHECK ADD  CONSTRAINT [Employee
lives in city] FOREIGN KEY([CityID])
REFERENCES [dbo].[Cities] ([CityID])
GO

ALTER TABLE [dbo].[Employees] CHECK CONSTRAINT [Employee lives in city]
GO

ALTER TABLE [dbo].[Employees]  WITH CHECK ADD  CONSTRAINT
[Employee's boss is] FOREIGN KEY([EmployeeID])
REFERENCES [dbo].[Employees] ([EmployeeID])
GO

ALTER TABLE [dbo].[Employees] CHECK CONSTRAINT [Employee's boss is]
GO

ALTER TABLE [dbo].[Employees]  WITH CHECK ADD  CONSTRAINT
[CK_Employees] CHECK  ((NOT [FirstName] like '[0-9]'))
GO
```

```
ALTER TABLE [dbo].[Employees] CHECK CONSTRAINT [CK_Employees]
GO

ALTER TABLE [dbo].[Employees]  WITH CHECK ADD  CONSTRAINT
[CK_Employees_1] CHECK  ((NOT [LastName] like '[0-9]'))
GO

ALTER TABLE [dbo].[Employees] CHECK CONSTRAINT [CK_Employees_1]
GO

ALTER TABLE [dbo].[Employees]  WITH CHECK ADD  CONSTRAINT
[EmployeePhoneConstraint] CHECK  ((isnumeric([Phone])=(1)))
GO

ALTER TABLE [dbo].[Employees] CHECK CONSTRAINT
[EmployeePhoneConstraint]
GO
```

26. Cities - słownik miast

Klucz główny: CityID [typ int, nie może być nullem]
Klucz obcy: CountryID [typ int, nie może być nullem] (Klucz z tabeli Countries)
nazwa miasta: CityName [typ varchar(50), nie może być nullem]
kod pocztowy: PostalCode [typ varchar(6), nie może być nullem]
Warunki integralnościowe:

- PostalCode musi być w postaci XX-XXX, XXXXX, XXXXXX gdzie X to cyfra:
  CHECK (([PostalCode] like '[0-9][0-9]-[0-9][0-9][0-9]' OR [PostalCode] like replicate('[0-9]',(5)) OR [PostalCode] like replicate('[0-9]',(6))))

```
CREATE TABLE [dbo].[Cities](
	[CityID] [int] NOT NULL,
	[CountryID] [int] NOT NULL,
	[CityName] [varchar](50) NOT NULL,
	[PostalCode] [varchar](6) NOT NULL,
 CONSTRAINT [PK_Cities] PRIMARY KEY CLUSTERED
(
	[CityID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON
[PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Cities]  WITH CHECK ADD  CONSTRAINT [City is in
Country] FOREIGN KEY([CountryID])
REFERENCES [dbo].[Countries] ([CountryID])
ON UPDATE CASCADE
ON DELETE CASCADE
```

```
GO

ALTER TABLE [dbo].[Cities] CHECK CONSTRAINT [City is in Country]
GO

ALTER TABLE [dbo].[Cities]  WITH CHECK ADD  CONSTRAINT
[PostalCodeConstraint] CHECK  (([PostalCode] like '[0-9][0-9]-[0-9][0-9][0-9]' OR
[PostalCode] like replicate('[0-9]',(5)) OR [PostalCode] like replicate('[0-9]',(6))))
GO

ALTER TABLE [dbo].[Cities] CHECK CONSTRAINT [PostalCodeConstraint]
GO
```

27. **Countries** - słownik krajów
    Klucz główny: CountryID [typ int, nie może być nullem]
    nazwa kraju: CountryName [typ varchar(50), nie może być nullem]
    Warunki integralnościowe:
        - CountryName musi być unikalne: *CHECK(UNIQUE([CountryName]))*

```
CREATE TABLE [dbo].[Countries](
	[CountryID] [int] NOT NULL,
	[CountryName] [varchar](50) NOT NULL,
 CONSTRAINT [PK_Countries] PRIMARY KEY CLUSTERED
(
	[CountryID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON
[PRIMARY],
 CONSTRAINT [IX_Countries] UNIQUE NONCLUSTERED
(
	[CountryName] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON
[PRIMARY]
) ON [PRIMARY]
GO
```

28. **ReservationParameters** - reprezentacja parametrów do rezerwacji dla klientów indywidualnych
    Klucz główny: ParameterID [typ int, nie może być nullem]
    wartość zmiennej: Value [typ decimal (18,2), nie może być nullem]
    nazwa zmiennej: ParameterName [typ varchar(50), nie może być nullem]
    Warunki integralnościowe:

- Value nie może być ujemne: *CHECK (([Value]>=(0)))*

```
CREATE TABLE [dbo].[ReservationParameters](
	[ParameterID] [int] NOT NULL,
	[ParameterName] [varchar](50) NOT NULL,
	[Value] [decimal](18, 2) NOT NULL,
 CONSTRAINT [PK_ReservationParameters] PRIMARY KEY CLUSTERED
(
	[ParameterID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON
[PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[ReservationParameters]  WITH CHECK ADD  CONSTRAINT
[CK_ReservationParameters] CHECK  (([Value]>=(0)))
GO

ALTER TABLE [dbo].[ReservationParameters] CHECK CONSTRAINT
[CK_ReservationParameters]
GO
```

Widoki

1. **AllEmployeesView** - widok na wszystkich pracowników i ich dane

```
CREATE VIEW [dbo].[AllEmployeesView]
AS
SELECT dbo.Employees.LastName, dbo.Employees.Phone, dbo.Employees.FirstName,
dbo.Cities.CityName, dbo.Cities.PostalCode, dbo.Countries.CountryName
FROM    dbo.Employees INNER JOIN
             dbo.Cities ON dbo.Employees.CityID = dbo.Cities.CityID INNER JOIN
             dbo.Countries ON dbo.Cities.CountryID = dbo.Countries.CountryID
GO
```

2. **AllReservedTablesView** - widok na wszystkie zarezerwowane stoliki wraz z datami rozpoczęcia i zakończenia rezerwacji

```
CREATE VIEW [dbo].[AllReservedTablesView]
AS
```

```
SELECT dbo.Reservations.StartDate, dbo.Reservations.EndDate, dbo.Tables.TableID,
dbo.Tables.SeatsAmount
FROM     dbo.CompanyReservations INNER JOIN
               dbo.CompanyReservationDetails ON dbo.CompanyReservations.ReservationID
= dbo.CompanyReservationDetails.ReservationID INNER JOIN
               dbo.Tables ON dbo.CompanyReservationDetails.TableID = dbo.Tables.TableID
INNER JOIN
               dbo.Reservations ON dbo.CompanyReservations.ReservationID =
dbo.Reservations.ReservationID
WHERE dbo.CompanyReservationDetails.TableID is not NULL
UNION
SELECT dbo.Reservations.StartDate, dbo.Reservations.EndDate, dbo.Tables.TableID,
dbo.Tables.SeatsAmount
FROM     dbo.Reservations INNER JOIN
               dbo.IndividualReservations ON dbo.Reservations.ReservationID =
dbo.IndividualReservations.ReservationID INNER JOIN
               dbo.Tables ON dbo.IndividualReservations.TableID = dbo.Tables.TableID
WHERE dbo.IndividualReservations.TableID is not NULL



GO
```

3. **AllTablesView** - widok na wszystkie stoliki

```
CREATE VIEW [dbo].[AllTablesView]
AS
SELECT TableID, SeatsAmount
FROM     dbo.Tables
GO
```

4. **AmountOfDiscountsUsedView** - widok na ilość używanych zniżek w czasie

```
CREATE VIEW [dbo].[AmountOfDiscountsUsedView]
AS
SELECT       COUNT(*) AS Amount, YEAR(OrderedDate) AS Year, MONTH(OrderedDate)
AS Month, DATEPART(dw, OrderedDate) AS Day, DATEPART(hh, OrderedDate) AS Hour
FROM          dbo.OrdersWithCalculatedDiscount
WHERE       (DiscountedValue IS NOT NULL)
```

```
GROUP BY YEAR(OrderedDate), MONTH(OrderedDate), DATEPART(dw, OrderedDate),
DATEPART(hh, OrderedDate)
GO
```

5. **AmountOfMenuChangesView** - widok na ilość zmian menu w czasie

```
CREATE VIEW [dbo].[AmountOfMenuChangesView]
AS
SELECT StartTable.Year, StartTable.Month, StartTable.Amount AS AddedAmount,
EndTable.Amount AS RemovedAmount
FROM     (SELECT YEAR(StartDate) AS Year, MONTH(StartDate) AS Month, COUNT(*)
AS Amount
               FROM      dbo.Menu
               GROUP BY YEAR(StartDate), MONTH(StartDate)) AS StartTable INNER JOIN
                  (SELECT YEAR(EndDate) AS Year, MONTH(EndDate) AS Month, COUNT(*)
AS Amount
                   FROM      dbo.Menu AS Menu_1
                   GROUP BY YEAR(EndDate), MONTH(EndDate)) AS EndTable ON
StartTable.Year = EndTable.Year AND StartTable.Month = EndTable.Month
GO
```

6. **AmountOfServedOrdersByEmployeesView** - widok na ilość wydanych zamówień przez pracowników

```
CREATE VIEW [dbo].[AmountOfServedOrdersByEmployeesView]
AS
SELECT       dbo.Employees.FirstName, dbo.Employees.LastName,
dbo.Employees.Phone, dbo.Cities.CityName, dbo.Cities.PostalCode,
dbo.Countries.CountryName, YEAR(dbo.Orders.ServedDate) AS Year,
MONTH(dbo.Orders.ServedDate)
                    AS Month, COUNT(*) AS Amount
FROM          dbo.Employees INNER JOIN
                    dbo.Orders ON dbo.Employees.EmployeeID = dbo.Orders.EmployeeID
INNER JOIN
                    dbo.Cities ON dbo.Employees.CityID = dbo.Cities.CityID INNER JOIN
                    dbo.Countries ON dbo.Cities.CountryID = dbo.Countries.CountryID
WHERE       (dbo.Orders.ServedDate IS NOT NULL)
GROUP BY dbo.Employees.FirstName, dbo.Employees.LastName,
dbo.Employees.Phone, dbo.Cities.CityName, dbo.Cities.PostalCode,
```

```
dbo.Countries.CountryName, YEAR(dbo.Orders.ServedDate),
MONTH(dbo.Orders.ServedDate)
GO
```

7. **CompaniesOrdersAmountView** - widok na ilość zamówień dokonanych przez firmy

```
CREATE VIEW [dbo].[CompaniesOrdersAmountView]
AS
SELECT COUNT(*) AS Amount, YEAR(dbo.Orders.OrderedDate) AS Year,
MONTH(dbo.Orders.OrderedDate) AS Month, DATEPART(dw, dbo.Orders.OrderedDate)
AS Day, DATEPART(hh, dbo.Orders.OrderedDate) AS Hour
FROM     dbo.Orders INNER JOIN
              dbo.Clients ON dbo.Clients.ClientID = dbo.Orders.ClientID INNER JOIN
              dbo.Companies ON dbo.Companies.ClientID = dbo.Clients.ClientID
GROUP BY YEAR(dbo.Orders.OrderedDate), MONTH(dbo.Orders.OrderedDate),
DATEPART(dw, dbo.Orders.OrderedDate), DATEPART(hh, dbo.Orders.OrderedDate)
GO
```

8. **CompaniesOrdersCostsView** - widok na koszt zamówień dokonanych przez firmy

```
CREATE VIEW [dbo].[CompaniesOrdersCostsView]
AS
SELECT       YEAR(dbo.OrdersTotalCostsView.OrderedDate) AS Year,
MONTH(dbo.OrdersTotalCostsView.OrderedDate) AS Month, DATEPART(d,
dbo.OrdersTotalCostsView.OrderedDate) AS Day, DATEPART(hh,
                  dbo.OrdersTotalCostsView.OrderedDate) AS Hour,
SUM(dbo.OrdersTotalCostsView.TotalCost) AS TotalCost,
SUM(dbo.OrdersTotalCostsView.DiscountedCost) AS DiscountedCost
FROM          dbo.OrdersTotalCostsView INNER JOIN
                  dbo.Orders ON dbo.Orders.OrderID = dbo.OrdersTotalCostsView.OrderID
INNER JOIN
                  dbo.Clients ON dbo.Orders.ClientID = dbo.Clients.ClientID INNER JOIN
                  dbo.Companies ON dbo.Clients.ClientID = dbo.Companies.ClientID
GROUP BY YEAR(dbo.OrdersTotalCostsView.OrderedDate),
MONTH(dbo.OrdersTotalCostsView.OrderedDate), DATEPART(d,
```

```
dbo.OrdersTotalCostsView.OrderedDate), DATEPART(hh,
dbo.OrdersTotalCostsView.OrderedDate)
GO
```

9. **CompanyReservationsAmountView** - widok na ilość rezerwacji firm w czasie z podziałem na miesiące i lata

```
CREATE VIEW [dbo].[CompanyReservationsAmountView]
AS
SELECT       COUNT(*) AS Amount, YEAR(dbo.Reservations.StartDate) AS Year,
MONTH(dbo.Reservations.StartDate) AS Month, DATEPART(d,
dbo.Reservations.StartDate) AS Day, DATEPART(hh, dbo.Reservations.StartDate) AS
Hour
FROM         dbo.Reservations INNER JOIN
                  dbo.CompanyReservations ON dbo.CompanyReservations.ReservationID
= dbo.Reservations.ReservationID
GROUP BY MONTH(dbo.Reservations.StartDate), YEAR(dbo.Reservations.StartDate),
DATEPART(d, dbo.Reservations.StartDate), DATEPART(hh, dbo.Reservations.StartDate)
GO
```

10. **CompanyReservationsWithoutTablesView** - widok na rezerwacje bez przypisanych stolików

```
CREATE VIEW [dbo].[CompanyReservationsWithoutTablesView]
AS
SELECT       C.CompanyName, CRD.AmountOfPeople
FROM         dbo.Companies AS C INNER JOIN
                  dbo.CompanyReservations AS CR ON CR.ClientID = C.ClientID INNER
JOIN
                  dbo.CompanyReservationDetails AS CRD ON CRD.ReservationID =
CR.ReservationID INNER JOIN
                  dbo.Reservations ON dbo.Reservations.ReservationID =
CRD.ReservationID
WHERE        (ISNULL(CRD.TableID, 0) = 0) AND dbo.Reservations.StatusCode != 1
GO
```

11. **CompanyReservationTablesView** - widok na rezerwacje z przypisanymi stolikami z podziałem na miesiące i lata

```
CREATE VIEW [dbo].[CompanyReservationTablesView]
AS
SELECT COUNT(*) AS Amount, YEAR(dbo.Reservations.StartDate) AS Year,
MONTH(dbo.Reservations.StartDate) AS Month, DATEPART(dw,
dbo.Reservations.StartDate) AS Day, DATEPART(hh, dbo.Reservations.StartDate) AS
Hour
FROM     dbo.CompanyReservationDetails INNER JOIN
                  dbo.CompanyReservations ON dbo.CompanyReservationDetails.ReservationID
= dbo.CompanyReservations.ReservationID INNER JOIN
                  dbo.Reservations ON dbo.CompanyReservations.ReservationID =
dbo.Reservations.ReservationID
WHERE  (dbo.CompanyReservationDetails.TableID IS NOT NULL)
GROUP BY MONTH(dbo.Reservations.StartDate), YEAR(dbo.Reservations.StartDate),
DATEPART(dw, dbo.Reservations.StartDate), DATEPART(hh, dbo.Reservations.StartDate)
GO
```

12. **MonthlyAmountOfDiscountsUsedView** - widok na ilość używanych zniżek w czasie z podziałem na miesiące i lata

```
CREATE VIEW [dbo].[MonthlyAmountOfDiscountsView]
AS
SELECT Year, Month, SUM(Amount) AS Expr1
FROM     dbo.AmountOfDiscountsUsedView
GROUP BY Year, Month
GO
```

13. **DailyAmountOfDiscountsUsedView** - widok na ilość używanych zniżek w czasie z podziałem na dni tygodnia

```
CREATE VIEW [dbo].[DailyAmountOfDiscountsUsedView]
AS
SELECT Day, SUM(Amount) AS Expr1
FROM     dbo.AmountOfDiscountsUsedView
GROUP BY Day
GO
```

14. **HourlyAmountOfDiscountsUsedView** - widok na ilość używanych zniżek w czasie z podziałem na godziny

```
CREATE VIEW [dbo].[HourlyAmountOfDiscountsUsedView]
AS
SELECT Hour, SUM(Amount) AS Expr1
FROM     dbo.AmountOfDiscountsUsedView
GROUP BY Hour
GO
```

15. **AmountOfServedOrdersByEmployeesView** - widok na ilość obsłużonych zamówień przez każdego z pracowników w czasie

```
CREATE VIEW [dbo].[AmountOfServedOrdersByEmployeesView]
AS
SELECT       dbo.Employees.FirstName, dbo.Employees.LastName,
dbo.Employees.Phone, dbo.Cities.CityName, dbo.Cities.PostalCode,
dbo.Countries.CountryName, YEAR(dbo.Orders.ServedDate) AS Year,
MONTH(dbo.Orders.ServedDate)
                    AS Month, COUNT(*) AS Amount
FROM          dbo.Employees INNER JOIN
                    dbo.Orders ON dbo.Employees.EmployeeID = dbo.Orders.EmployeeID
INNER JOIN
                    dbo.Cities ON dbo.Employees.CityID = dbo.Cities.CityID INNER JOIN
                    dbo.Countries ON dbo.Cities.CountryID = dbo.Countries.CountryID
WHERE       (dbo.Orders.ServedDate IS NOT NULL)
GROUP BY dbo.Employees.FirstName, dbo.Employees.LastName,
dbo.Employees.Phone, dbo.Cities.CityName, dbo.Cities.PostalCode,
dbo.Countries.CountryName, YEAR(dbo.Orders.ServedDate),
MONTH(dbo.Orders.ServedDate)
GO
```

16. **CompaniesOrdersAmountView** - widok na ilość zamówień złożonych przez firmy w czasie

```
CREATE VIEW [dbo].[CompaniesOrdersAmountView]
AS
SELECT COUNT(*) AS Amount, YEAR(dbo.Orders.OrderedDate) AS Year,
MONTH(dbo.Orders.OrderedDate) AS Month, DATEPART(dw, dbo.Orders.OrderedDate)
AS Day, DATEPART(hh, dbo.Orders.OrderedDate) AS Hour
FROM     dbo.Orders INNER JOIN
              dbo.Clients ON dbo.Clients.ClientID = dbo.Orders.ClientID INNER JOIN
```

```
        dbo.Companies ON dbo.Companies.ClientID = dbo.Clients.ClientID
GROUP BY YEAR(dbo.Orders.OrderedDate), MONTH(dbo.Orders.OrderedDate),
DATEPART(dw, dbo.Orders.OrderedDate), DATEPART(hh, dbo.Orders.OrderedDate)
GO
```

17. **MonthlyCompaniesOrdersAmountView** - widok na ilość zamówień złożonych przez firmy w czasie z podziałem na miesiące i lata

```
CREATE VIEW [dbo].[MonthlyCompaniesOrdersAmountView]
AS
SELECT Year, Month, SUM(Amount) AS Amount
FROM    dbo.CompaniesOrdersAmountView
GROUP BY Year, Month
GO
```

18. **DailyCompaniesOrdersAmountView** - widok na ilość zamówień złożonych przez firmy w czasie z podziałem na dni tygodnia

```
CREATE VIEW [dbo].[DailyCompaniesOrdersAmountView]
AS
SELECT Day, SUM(Amount) AS Amount
FROM    dbo.CompaniesOrdersAmountView
GROUP BY Day
GO
```

19. **HourlyCompaniesOrdersAmountView** - widok na ilość zamówień złożonych przez firmy w czasie z podziałem na godziny

```
CREATE VIEW [dbo].[HourlyCompaniesOrdersAmountView]
AS
SELECT Hour, SUM(Amount) AS Amount
FROM    dbo.CompaniesOrdersAmountView
GROUP BY Hour
GO
```

20. **CompaniesOrdersCostsView** - widok na łączną kwotę zamówień składanych przez firmy w czasie

```
CREATE VIEW [dbo].[CompaniesOrdersCostsView]
AS
SELECT       YEAR(dbo.OrdersTotalCostsView.OrderedDate) AS Year,
MONTH(dbo.OrdersTotalCostsView.OrderedDate) AS Month, DATEPART(d,
dbo.OrdersTotalCostsView.OrderedDate) AS Day, DATEPART(hh,
                  dbo.OrdersTotalCostsView.OrderedDate) AS Hour,
SUM(dbo.OrdersTotalCostsView.TotalCost) AS TotalCost,
SUM(dbo.OrdersTotalCostsView.DiscountedCost) AS DiscountedCost
FROM          dbo.OrdersTotalCostsView INNER JOIN
                  dbo.Orders ON dbo.Orders.OrderID = dbo.OrdersTotalCostsView.OrderID
INNER JOIN
                  dbo.Clients ON dbo.Orders.ClientID = dbo.Clients.ClientID INNER JOIN
                  dbo.Companies ON dbo.Clients.ClientID = dbo.Companies.ClientID
GROUP BY YEAR(dbo.OrdersTotalCostsView.OrderedDate),
MONTH(dbo.OrdersTotalCostsView.OrderedDate), DATEPART(d,
dbo.OrdersTotalCostsView.OrderedDate), DATEPART(hh,
dbo.OrdersTotalCostsView.OrderedDate)
GO
```

21. **MonthlyCompaniesOrdersCostsView** - widok na łączną kwotę zamówień składanych przez firmy w czasie z podziałem na miesiące i lata

```
CREATE VIEW [dbo].[MonthlyCompaniesOrdersCostsView]
AS
SELECT Year, Month, SUM(TotalCost) AS TotalCost, SUM(DiscountedCost) AS
DiscountedCost
FROM     dbo.CompaniesOrdersCosts
GROUP BY Year, Month
GO
```

22. **DailyCompaniesOrdersCostsView** - widok na łączną kwotę zamówień składanych przez firmy w czasie z podziałem na dni tygodnia

```
CREATE VIEW [dbo].[DailyCompaniesOrdersCostsView]
AS
SELECT Day, SUM(TotalCost) AS TotalCost, SUM(DiscountedCost) AS DiscountedCost
```

```
FROM     dbo.CompaniesOrdersCosts
GROUP BY Day
GO
```

23. **HourlyCompaniesOrdersCostsView** - widok na łączną kwotę zamówień składanych przez firmy w czasie z podziałem na godziny

```
CREATE VIEW [dbo].[HourlyCompaniesOrdersCostsView]
AS
SELECT Hour, SUM(TotalCost) AS TotalCost, SUM(DiscountedCost) AS DiscountedCost
FROM     dbo.CompaniesOrdersCosts
GROUP BY Hour
GO
```

24. **CompanyReservationsAmountView** - widok na łączną ilość rezerwacji dla firm w czasie

```
CREATE VIEW [dbo].[CompanyReservationsAmountView]
AS
SELECT       COUNT(*) AS Amount, YEAR(dbo.Reservations.StartDate) AS Year,
MONTH(dbo.Reservations.StartDate) AS Month, DATEPART(d,
dbo.Reservations.StartDate) AS Day, DATEPART(hh, dbo.Reservations.StartDate) AS
Hour
FROM          dbo.Reservations INNER JOIN
                  dbo.CompanyReservations ON dbo.CompanyReservations.ReservationID
= dbo.Reservations.ReservationID
GROUP BY MONTH(dbo.Reservations.StartDate), YEAR(dbo.Reservations.StartDate),
DATEPART(d, dbo.Reservations.StartDate), DATEPART(hh, dbo.Reservations.StartDate)
GO
```

25. **MonthlyCompanyReservationsAmountView** - widok na łączną ilość rezerwacji dla firm w czasie z podziałem na miesiące

```
CREATE VIEW [dbo].[MonthlyCompanyReservationsAmountView]
AS
SELECT Year, Month, SUM(Amount) AS Amount
FROM     dbo.CompanyReservationsAmountView
GROUP BY Year, Month
GO
```

26. **DailyCompanyReservationsAmountView** - widok na łączną ilość rezerwacji dla firm w czasie z podziałem na dni tygodnia

```
CREATE VIEW [dbo].[DailyCompanyReservationsAmountView]
AS
SELECT Day, SUM(Amount) AS Amount
FROM     dbo.CompanyReservationsAmountView
GROUP BY Day
GO
```

27. **HourlyCompanyReservationsAmountView** - widok na łączną ilość rezerwacji dla firm w czasie z podziałem na godziny

```
CREATE VIEW [dbo].[HourlyCompanyReservationsAmountView]
AS
SELECT Hour, SUM(Amount) AS Amount
FROM     dbo.CompanyReservationsAmountView
GROUP BY Hour
GO
```

28. **CompanyReservationTablesView**- widok na łączną ilość zarezerwowanych stolików dla firm w czasie

```
CREATE VIEW [dbo].[CompanyReservationTablesView]
AS
SELECT COUNT(*) AS Amount, YEAR(dbo.Reservations.StartDate) AS Year,
MONTH(dbo.Reservations.StartDate) AS Month, DATEPART(dw,
dbo.Reservations.StartDate) AS Day, DATEPART(hh, dbo.Reservations.StartDate) AS
Hour
FROM     dbo.CompanyReservationDetails INNER JOIN
               dbo.CompanyReservations ON dbo.CompanyReservationDetails.ReservationID
= dbo.CompanyReservations.ReservationID INNER JOIN
               dbo.Reservations ON dbo.CompanyReservations.ReservationID =
dbo.Reservations.ReservationID
WHERE  (dbo.CompanyReservationDetails.TableID IS NOT NULL)
GROUP BY MONTH(dbo.Reservations.StartDate), YEAR(dbo.Reservations.StartDate),
DATEPART(dw, dbo.Reservations.StartDate), DATEPART(hh, dbo.Reservations.StartDate)
GO
```

29. **MonthlyCompanyReservationTablesView** - widok na łączną ilość zarezerwowanych stolików dla firm w czasie z podziałem na miesiące

```
CREATE VIEW [dbo].[MonthlyCompanyReservationTablesView]
AS
SELECT Year, Month, SUM(Amount) AS Amount
FROM     dbo.CompanyReservationTablesView
GROUP BY Year, Month
GO
```

30. **DailyCompanyReservationTablesView** - widok na łączną ilość zarezerwowanych stolików dla firm w czasie z podziałem na dni tygodnia

```
CREATE VIEW [dbo].[DailyCompanyReservationTablesView]
AS
SELECT Day, SUM(Amount) AS Amount
FROM     dbo.CompanyReservationTablesView
GROUP BY Day
GO
```

31. **HourlyCompanyReservationTablesView** - widok na łączną ilość zarezerwowanych stolików dla firm w czasie z podziałem na godziny

```
CREATE VIEW [dbo].[HourlyCompanyReservationTablesView]
AS
SELECT Hour, SUM(Amount) AS Amount
FROM     dbo.CompanyReservationTablesView
GROUP BY Hour
GO
```

32. **DiscountsSavedMoneyView** - widok na kwoty oszczędzone przez klientów dzięki zniżkom w czasie

```
CREATE VIEW [dbo].[DiscountsSavedMoneyView]
AS
SELECT       SUM(ISNULL(dbo.Orders.DiscountValue, 0) * (dbo.Menu.Price *
dbo.OrderDetails.Quantity)) AS Amount, YEAR(dbo.Orders.OrderedDate) AS Year,
MONTH(dbo.Orders.OrderedDate) AS Month, DATEPART(dw,
```

```
                dbo.Orders.OrderedDate) AS Day, DATEPART(hh,
dbo.Orders.OrderedDate) AS Hour
FROM        dbo.Orders INNER JOIN
                dbo.OrderDetails ON dbo.Orders.OrderID = dbo.OrderDetails.OrderID
INNER JOIN
                dbo.Menu ON dbo.OrderDetails.RecordID = dbo.Menu.RecordID
WHERE       (dbo.Orders.DiscountValue IS NOT NULL)
GROUP BY YEAR(dbo.Orders.OrderedDate), MONTH(dbo.Orders.OrderedDate),
DATEPART(dw, dbo.Orders.OrderedDate), DATEPART(hh, dbo.Orders.OrderedDate)
GO
```

33. **MonthlyDiscountsSavedMoneyView** - widok na kwoty oszczędzone przez klientów dzięki zniżkom w czasie z podziałem na miesiące i lata

```
CREATE VIEW [dbo].[MonthlyDiscountsSavedMoneyView]
AS
SELECT Year, Month, SUM(Amount) AS Expr1
FROM    dbo.DiscountsSavedMoneyView
GROUP BY Year, Month
GO
```

34. **DailyDiscountsSavedMoneyView**- widok na kwoty oszczędzone przez klientów dzięki zniżkom w czasie z podziałem na dni tygodnia

```
CREATE VIEW [dbo].[DailyDiscountsSavedMoneyView]
AS
SELECT Day, SUM(Amount) AS Amount
FROM    dbo.DiscountsSavedMoneyView
GROUP BY Day
GO
```

35. **HourlyDiscountsSavedMoneyView** - widok na kwoty oszczędzone przez klientów dzięki zniżkom w czasie z podziałem na godziny

```
CREATE VIEW [dbo].[HourlyDiscountsSavedMoneyView]
AS
SELECT Hour, SUM(Amount) AS Amount
FROM    dbo.DiscountsSavedMoneyView
GROUP BY Hour
GO
```

36. **IndividualClientOrdersCostsView** - widok na łączną kwotę zamówień dla klientów w czasie

```
CREATE VIEW [dbo].[IndividualClientOrdersCostsView]
AS
SELECT YEAR(dbo.OrdersTotalCostsView.OrderedDate) AS Year,
MONTH(dbo.OrdersTotalCostsView.OrderedDate) AS Month, DATEPART(dw,
dbo.OrdersTotalCostsView.OrderedDate) AS Day, DATEPART(hh,
            dbo.OrdersTotalCostsView.OrderedDate) AS Hour,
SUM(dbo.OrdersTotalCostsView.TotalCost) AS TotalCost,
SUM(dbo.OrdersTotalCostsView.DiscountedCost) AS DiscountedCost
FROM    dbo.OrdersTotalCostsView INNER JOIN
            dbo.Orders ON dbo.Orders.OrderID = dbo.OrdersTotalCostsView.OrderID
INNER JOIN
            dbo.Clients ON dbo.Orders.ClientID = dbo.Clients.ClientID INNER JOIN
            dbo.IndividualClients ON dbo.Clients.ClientID = dbo.IndividualClients.ClientID
GROUP BY YEAR(dbo.OrdersTotalCostsView.OrderedDate),
MONTH(dbo.OrdersTotalCostsView.OrderedDate), DATEPART(dw,
dbo.OrdersTotalCostsView.OrderedDate), DATEPART(hh,
dbo.OrdersTotalCostsView.OrderedDate)
GO
```

37. **MonthlyIndividualClientOrdersCostsView** - widok na łączną kwotę zamówień dla klientów w czasie z podziałem na miesiące i lata

```
CREATE VIEW [dbo].[MonthlyIndividualClientOrdersCoststView]
AS
SELECT Year, Month, SUM(TotalCost) AS TotalCost, SUM(DiscountedCost) AS
DiscountedCost
FROM    dbo.IndividualClientOrdersCostsView
GROUP BY Year, Month
GO
```

38. **DailyIndividualClientOrdersCostsView** - widok na łączną kwotę zamówień dla klientów w czasie z podziałem na dni tygodnia

```
CREATE VIEW [dbo].[DailyIndividualClientsOrdersCostsView]
AS
SELECT Day, SUM(TotalCost) AS TotalCost, SUM(DiscountedCost) AS DiscountedCost
FROM    dbo.IndividualClientOrdersCostsView
```

```
GROUP BY Day
GO
```

39. **HourlyIndividualClientOrdersCostsView** - widok na łączną kwotę zamówień dla klientów w czasie z podziałem na godziny

```
CREATE VIEW [dbo].[HourlyIndividualClientsOrdersCostsView]
AS
SELECT Hour, SUM(TotalCost) AS TotalCost, SUM(DiscountedCost) AS DiscountedCost
FROM     dbo.IndividualClientOrdersCostsView
GROUP BY Hour
GO
```

40. **IndividualClientsOrdersAmountView** - widok na łączną ilość zamówień dla klientów w czasie

```
CREATE VIEW [dbo].[IndividualClientsOrdersAmountView]
AS
SELECT COUNT(*) AS Amount, YEAR(dbo.Orders.OrderedDate) AS year,
MONTH(dbo.Orders.OrderedDate) AS Month, DATEPART(dw, dbo.Orders.OrderedDate)
AS Day, DATEPART(hh, dbo.Orders.OrderedDate) AS Hour
FROM     dbo.Orders INNER JOIN
              dbo.Clients ON dbo.Clients.ClientID = dbo.Orders.ClientID INNER JOIN
              dbo.IndividualClients ON dbo.IndividualClients.ClientID = dbo.Clients.ClientID
GROUP BY YEAR(dbo.Orders.OrderedDate), MONTH(dbo.Orders.OrderedDate),
DATEPART(dw, dbo.Orders.OrderedDate), DATEPART(hh, dbo.Orders.OrderedDate)
GO
```

41. **MonthlyIndividualClientsOrdersAmountView** - widok na łączną ilość zamówień dla klientów w czasie z podziałem na miesiące i lata

```
CREATE VIEW [dbo].[MonthlyIndividualClientsOrdersAmountView]
AS
SELECT year, Month, SUM(Amount) AS Amount
FROM     dbo.IndividualClientsOrdersAmountView
GROUP BY year, Month
GO
```

42. **DailyIndividualClientsOrdersAmountView** - widok na łączną ilość zamówień dla klientów w czasie z podziałem na dni tygodnia

```
CREATE VIEW [dbo].[DailyIndividualClientsOrdersAmountView]
AS
SELECT Day, SUM(Amount) AS Amount
FROM     dbo.IndividualClientsOrdersAmountView
GROUP BY Day
GO
```

43. **HourlyIndividualClientsOrdersAmountView** - widok na łączną ilość zamówień dla klientów w czasie z podziałem na godziny

```
CREATE VIEW [dbo].[HourlyIndividualClientsOrdersAmountView]
AS
SELECT Hour, SUM(Amount) AS Amount
FROM     dbo.IndividualClientsOrdersAmountView
GROUP BY Hour
GO
```

44. **IndividualReservationsAmountView** - widok na łączną ilość rezerwacji dla klientów indywidualnych w czasie

```
CREATE VIEW [dbo].[IndividualReservationsAmountView]
AS
SELECT COUNT(*) AS Amount, YEAR(dbo.Reservations.StartDate) AS Year,
MONTH(dbo.Reservations.StartDate) AS Month, DATEPART(dw,
dbo.Reservations.StartDate) AS Day, DATEPART(hh, dbo.Reservations.StartDate) AS
Hour
FROM     dbo.Reservations INNER JOIN
               dbo.IndividualReservations ON dbo.IndividualReservations.ReservationID =
dbo.Reservations.ReservationID
GROUP BY MONTH(dbo.Reservations.StartDate), YEAR(dbo.Reservations.StartDate),
DATEPART(dw, dbo.Reservations.StartDate), DATEPART(hh, dbo.Reservations.StartDate)
GO
```

45. **MonthlyIndividualReservationsAmountView** - widok na łączną ilość rezerwacji dla klientów indywidualnych w czasie z podziałem na miesiące i lata

```
CREATE VIEW [dbo].[MonthlyIndividualClientsReservationsAmountView]
AS
SELECT Year, Month, SUM(Amount) AS Amount
FROM     dbo.IndividualReservationsAmountView
GROUP BY Year, Month
GO
```

46. **DailyIndividualReservationsAmountView** - widok na łączną ilość rezerwacji dla klientów indywidualnych w czasie z podziałem na dni tygodnia

```
CREATE VIEW [dbo].[DailyIndividualReservationsAmountView]
AS
SELECT Day, SUM(Amount) AS Expr1
FROM     dbo.IndividualReservationsAmountView
GROUP BY Day
GO
```

47. **HourlyIndividualClientsOrdersAmountView** - widok na łączną ilość rezerwacji dla klientów indywidualnych w czasie z podziałem na godziny

```
CREATE VIEW [dbo].[HourlyIndividualReservationsAmountView]
AS
SELECT Hour, SUM(Amount) AS Expr1
FROM     dbo.IndividualReservationsAmountView
GROUP BY Hour
GO
```

48. **IndividualReservationTablesView** - widok na łączną ilość zarezerwowanych stolików dla klientów indywidualnych w czasie

```
CREATE VIEW [dbo].[IndividualReservationTablesView]
AS
SELECT COUNT(*) AS Amount, YEAR(dbo.Reservations.StartDate) AS Year,
MONTH(dbo.Reservations.StartDate) AS Month, DATEPART(dw,
dbo.Reservations.StartDate) AS Day, DATEPART(hh, dbo.Reservations.StartDate) AS
Hour
FROM     dbo.IndividualReservations INNER JOIN
```

```
            dbo.Reservations ON dbo.IndividualReservations.ReservationID =
dbo.Reservations.ReservationID
WHERE  (dbo.IndividualReservations.TableID IS NOT NULL)
GROUP BY MONTH(dbo.Reservations.StartDate), YEAR(dbo.Reservations.StartDate),
DATEPART(dw, dbo.Reservations.StartDate), DATEPART(hh, dbo.Reservations.StartDate)
GO
```

49. **IndividualReservationsWithoutTablesView** - widok na na rezerwacje dla klientów indywidualnych bez przypisanych stolików

```
CREATE VIEW [dbo].[IndividualReservationsWithoutTablesView]
AS
SELECT       IC.FirstName, IC.LastName, IR.AmountOfPeople
FROM         dbo.IndividualReservations AS IR INNER JOIN
               dbo.IndividualClients AS IC ON IR.ClientID = IC.ClientID INNER JOIN
               dbo.Reservations ON dbo.Reservations.ReservationID = IR.ReservationID
WHERE       (ISNULL(IR.TableID, 0) = 0) AND (dbo.Reservations.StatusCode != 1)
GO
```

50. **IndividualNumberOfOrdersView** - widok na łączną ilość zarezerwowanych stolików dla klientów indywidualnych

```
CREATE VIEW [dbo].[IndividualNumberOfOrdersView]
AS
SELECT       O.ClientID, ISNULL(COUNT(*),0) AS NumberOfOrders
FROM         Orders AS O
inner join IndividualClients as IC on O.ClientID = IC.ClientID
GROUP BY O.ClientID
GO
```

51. **MonthlyIndividualReservationTablesView** - widok na łączną ilość zarezerwowanych stolików dla klientów indywidualnych w czasie z podziałem na miesiące i lata

```
CREATE VIEW [dbo].[MonthlyIndividualReservationTablesView]
AS
SELECT Year, Month, SUM(Amount) AS Expr1
FROM     dbo.IndividualReservationTablesView
```

```
GROUP BY Year, Month
GO
```

52. DailyIndividualReservationTablesView - widok na łączną ilość zarezerwowanych stolików dla klientów indywidualnych w czasie z podziałem na dni tygodnia

```
CREATE VIEW [dbo].[DailyIndividualReservationTablesView]
AS
SELECT Day, SUM(Amount) AS Expr1
FROM     dbo.IndividualReservationTablesView
GROUP BY Day
GO
```

53. HourlyIndividualReservationTablesView - widok na łączną ilość zarezerwowanych stolików dla klientów indywidualnych w czasie z podziałem na godziny

```
CREATE VIEW [dbo].[HourlyIndividualReservationTablesView]
AS
SELECT Hour, SUM(Amount) AS Expr1
FROM     dbo.IndividualReservationTablesView
GROUP BY Hour
GO
```

54. MenuView - widok na aktualne menu

```
CREATE VIEW [dbo].[MenuView]
AS
SELECT dbo.Menu.Price, dbo.Meals.MealName, dbo.Categories.CategoryName,
dbo.Categories.Description
FROM     dbo.Menu INNER JOIN
               dbo.Meals ON dbo.Menu.MealID = dbo.Meals.MealID INNER JOIN
               dbo.Categories ON dbo.Meals.CategoryID = dbo.Categories.CategoryID
WHERE  (dbo.Menu.StartDate <= GETDATE()) AND (GETDATE() <=
ISNULL(dbo.Menu.EndDate, GETDATE()))
GO
```

55. MenuWithRecordIDView - widok na aktualne menu z RecordID

```
CREATE VIEW [dbo].[MenuWithRecordIDView]
AS
SELECT       dbo.Menu.Price, dbo.Meals.MealName, dbo.Categories.CategoryName,
dbo.Categories.Description, dbo.Menu.RecordID
FROM         dbo.Menu INNER JOIN
                  dbo.Meals ON dbo.Menu.MealID = dbo.Meals.MealID INNER JOIN
                  dbo.Categories ON dbo.Meals.CategoryID = dbo.Categories.CategoryID
WHERE       (dbo.Menu.StartDate <= GETDATE()) AND (GETDATE() <=
ISNULL(dbo.Menu.EndDate, GETDATE()))
GO
```

56. OrdersAmountView - widok na łączną ilość zamówień w czasie

```
CREATE VIEW [dbo].[OrdersAmountView]
AS
SELECT COUNT(*) AS Amount, YEAR(OrderedDate) AS year, MONTH(OrderedDate) AS
Month, DATEPART(dw, OrderedDate) AS Day, DATEPART(hh, OrderedDate) AS Hour
FROM     dbo.Orders
GROUP BY YEAR(OrderedDate), MONTH(OrderedDate), DATEPART(dw, OrderedDate),
DATEPART(hh, OrderedDate)
GO
```

57. OrdersCoststView - widok na łączną kwotę zamówień w czasie

```
CREATE VIEW [dbo].[OrdersCostsView]
AS
SELECT YEAR(OrderedDate) AS Year, MONTH(OrderedDate) AS Month, DATEPART(dw,
OrderedDate) AS Day, DATEPART(hh, OrderedDate) AS Hour, SUM(TotalCost) AS
TotalCost, SUM(DiscountedCost) AS DiscountedCost
FROM     dbo.OrdersTotalCostsView
GROUP BY YEAR(OrderedDate), MONTH(OrderedDate), DATEPART(dw, OrderedDate),
DATEPART(hh, OrderedDate)
GO
```

58. OrdersTotalCoststView - pomocniczy widok, który dla każdego zamówienia przypisuje obliczony koszt zamówienia przed i po zniżce

```
CREATE VIEW [dbo].[OrdersTotalCostsView]
```

```
AS
SELECT dbo.Orders.OrderID, SUM(dbo.Menu.Price * dbo.OrderDetails.Quantity) AS
TotalCost, ROUND(SUM((dbo.Menu.Price * dbo.OrderDetails.Quantity) * (1 -
ISNULL(dbo.Orders.DiscountValue, 0))), 2) AS DiscountedCost,
            dbo.Orders.DiscountValue, dbo.Orders.OrderedDate,
dbo.Orders.RequiredDate, dbo.Orders.ServedDate, dbo.Orders.PaidDate,
dbo.Orders.TakeOut, dbo.Orders.PayingLater
FROM    dbo.Orders INNER JOIN
            dbo.OrderDetails ON dbo.Orders.OrderID = dbo.OrderDetails.OrderID INNER
JOIN
            dbo.Menu ON dbo.OrderDetails.RecordID = dbo.Menu.RecordID
GROUP BY dbo.Orders.OrderID, dbo.Orders.OrderedDate, dbo.Orders.RequiredDate,
dbo.Orders.ServedDate, dbo.Orders.PaidDate, dbo.Orders.DiscountValue,
dbo.Orders.TakeOut, dbo.Orders.PayingLater
GO
```

59. ReservationsAmountView- widok na łączną ilość rezerwacji w czasie

```
CREATE VIEW [dbo].[ReservationsAmountView]
AS
SELECT COUNT(*) AS Amount, YEAR(StartDate) AS Year, MONTH(StartDate) AS Month,
DATEPART(dw, StartDate) AS Day, DATEPART(hh, StartDate) AS Hour
FROM    dbo.Reservations
GROUP BY MONTH(StartDate), YEAR(StartDate), DATEPART(dw, StartDate),
DATEPART(hh, StartDate)
GO
```

60. ReservationTablesView - widok na łączną ilość zarezerwowanych stolików w czasie

```
CREATE VIEW [dbo].[ReservationTablesView]
AS
SELECT SUM(Amount) AS Amount, Year, Month, Day, Hour
FROM    (SELECT Amount, Year, Month, Day, Hour
            FROM    dbo.CompanyReservationTablesView
            UNION
            SELECT Amount, Year, Month, Day, Hour
            FROM    dbo.IndividualReservationTablesView) AS TotalTablesReservations
GROUP BY Year, Month, Day, Hour
GO
```

61. ReservationWithoutTablesView - widok na rezerwacje bez przypisanych stolików

```
CREATE VIEW [dbo].[ReservationsWithoutTablesView]
AS
SELECT      FirstName + ' ' + LastName AS Name, AmountOfPeople
FROM        IndividualReservationsWithoutTablesView
UNION
SELECT      CompanyName AS Name, AmountOfPeople
FROM        CompanyReservationsWithoutTablesView
GO
```

62. **UnregisteredCustomersOrdersCostsView** - widok na łączną kwotę zamówień składanych przez niezarejestrowanych klientów w czasie

```
CREATE VIEW [dbo].[UnregisteredCustomersOrdersCostsView]
AS
SELECT YEAR(dbo.OrdersTotalCostsView.OrderedDate) AS Year,
MONTH(dbo.OrdersTotalCostsView.OrderedDate) AS Month, DATEPART(dw,
dbo.OrdersTotalCostsView.OrderedDate) AS Day, DATEPART(hh,
            dbo.OrdersTotalCostsView.OrderedDate) AS Hour,
SUM(dbo.OrdersTotalCostsView.TotalCost) AS TotalCost,
SUM(dbo.OrdersTotalCostsView.DiscountedCost) AS DiscountedCost
FROM    dbo.OrdersTotalCostsView INNER JOIN
            dbo.Orders ON dbo.OrdersTotalCostsView.OrderID = dbo.Orders.OrderID
WHERE  (dbo.Orders.ClientID IS NULL)
GROUP BY YEAR(dbo.OrdersTotalCostsView.OrderedDate),
MONTH(dbo.OrdersTotalCostsView.OrderedDate), DATEPART(dw,
dbo.OrdersTotalCostsView.OrderedDate), DATEPART(hh,
dbo.OrdersTotalCostsView.OrderedDate)
GO
```

63. **MonthlyUnregisteredCustomersOrdersCostsView** - widok na łączną kwotę zamówień składanych przez niezarejestrowanych klientów w czasie z podziałem na rok i miesiąc

```
CREATE VIEW [dbo].[MonthlyUnregisteredCustomersOrdersCostsView]
AS
SELECT Year, Month, SUM(TotalCost) AS TotalCost, SUM(DiscountedCost) AS
DiscountedCost
FROM    dbo.UnregisteredCustomersOrdersCostsView
GROUP BY Year, Month
```

```
GO
```

64. **DailyUnregisteredCustomersOrdersCostsView** - widok na łączną kwotę zamówień składanych przez niezarejestrowanych klientów w czasie z podziałem na dni tygodnia

```
CREATE VIEW [dbo].[DailyUnregisteredCustomersOrdersCostsView]
AS
SELECT Day, SUM(TotalCost) AS TotalCost, SUM(DiscountedCost) AS DiscountedCost
FROM     dbo.UnregisteredCustomersOrdersCostsView
GROUP BY Day
GO
```

65. **HourlyUnregisteredCustomersOrdersCostsView** - widok na łączną kwotę zamówień składanych przez niezarejestrowanych klientów w czasie z podziałem na godziny

```
CREATE VIEW [dbo].[HourlyUnregisteredCustomersOrdersCostsView]
AS
SELECT Hour, SUM(TotalCost) AS TotalCost, SUM(DiscountedCost) AS DiscountedCost
FROM     dbo.UnregisteredCustomersOrdersCostsView
GROUP BY Hour
GO
```

66. **PermanentDiscountsForClientsView** - widok na zniżki na zawsze dla każdego klienta

```
CREATE VIEW [dbo].[PermanentDiscountsForClientsView]
AS
select distinct C.ClientID,
PD.Valid,
(select PDC.Value from PermanentDiscountsConstants as PDC
where PDC.StartDate <= PD.CollectingStartDate and PDC.EndDate >
PD.CollectingStartDate and PDC.ParameterID = 3) as PermanentDiscountValue
from Orders as O
inner join Clients as C on C.ClientID = O.ClientID
inner join IndividualClients as IC on IC.ClientID = C.ClientID
inner join PermanentDiscounts as PD on PD.ClientID = IC.ClientID
where PD.Valid = 1
union
select distinct C.ClientID,
```

```
PD.Valid,
0 as PermanentDiscountValue
from Orders as O
inner join Clients as C on C.ClientID = O.ClientID
inner join IndividualClients as IC on IC.ClientID = C.ClientID
inner join PermanentDiscounts as PD on PD.ClientID = IC.ClientID
where PD.Valid = 0
GO
```

67. **WaitingCompanyReservationsView** - widok na czekające na zatwierdzenie rezerwacje dla firm

```
CREATE VIEW [dbo].[WaitingCompanyReservationsView]
AS
SELECT dbo.CompanyReservations.OrderID,
dbo.CompanyReservationDetails.AmountOfPeople, dbo.Reservations.StartDate,
dbo.Reservations.EndDate
FROM     dbo.CompanyReservations INNER JOIN
              dbo.CompanyReservationDetails ON dbo.CompanyReservations.ReservationID
= dbo.CompanyReservationDetails.ReservationID INNER JOIN
              dbo.Reservations ON dbo.CompanyReservations.ReservationID =
dbo.Reservations.ReservationID INNER JOIN
              dbo.ReservationStatuses ON dbo.Reservations.StatusCode =
dbo.ReservationStatuses.StatusCode
WHERE  (dbo.ReservationStatuses.Description = 'Waiting')
GO
```

68. **OrdersWithCalculatedDiscount** - widok na łączną kwotę zamówień z obliczoną zniżką składanych przez klienta w czasie z podziałem na godziny

```
CREATE VIEW [dbo].[OrdersWithCalculatedDiscount]
AS
SELECT       OrderID, EmployeeID, OrderedDate, RequiredDate, ServedDate, PaidDate,
ClientID, TakeOut, PayingLater, (CASE WHEN DiscountID IN
                    (SELECT       DiscountID
                     FROM          OneTimeDiscounts) THEN
                    (SELECT       Value
                     FROM          OneTimeDiscountsConstants INNER JOIN
```

```
                                        OneTimeDiscountParameters ON
OneTimeDiscountParameters.ParameterID = OneTimeDiscountsConstants.ParameterID
                    WHERE       StartDate <
                                        (SELECT       CollectingStartDate
                                         FROM          Discounts
                                         WHERE       Discounts.DiscountID =
Orders.DiscountID) AND
                                        (SELECT       CollectingStartDate
                                         FROM          Discounts
                                         WHERE       Discounts.DiscountID =
Orders.DiscountID) <= ISNULL(EndDate, GETDATE()) AND ParameterName = 'R2')
WHEN DiscountID IN
                    (SELECT       DiscountID
                     FROM          PermanentDiscounts) THEN
                    (SELECT       Value
                     FROM          PermanentDiscountsConstants INNER JOIN
                                      PermamentDiscountsParameters ON
PermamentDiscountsParameters.ParameterID =
PermanentDiscountsConstants.ParameterID
                    WHERE       StartDate <
                                        (SELECT       CollectingStartDate
                                         FROM          Discounts
                                         WHERE       Discounts.DiscountID =
Orders.DiscountID) AND
                                        (SELECT       CollectingStartDate
                                         FROM          Discounts
                                         WHERE       Discounts.DiscountID =
Orders.DiscountID) <= ISNULL(EndDate, GETDATE()) AND ParameterName = 'R1') ELSE
0 END) AS DiscountedValue
FROM          dbo.Orders
GO
```

69. ClientsWithPermanentAndOneTimeDiscountView - widok na klientów z ważnymi obiema rodzajami zniżek

```
CREATE VIEW [dbo].[ClientsWithPermanentAndOneTimeDiscountView]
AS
select ClientID, DiscountID, 'Permanent Discount' as TypeOfDiscount from
PermanentDiscounts
where ClientID in (select ClientID from OneTimeDiscounts as OTD
                                        inner join Discounts as D on D.DiscountID =
OTD.DiscountID
```

```
                                        where D.Valid = 1 and OTD.ClientID in (select
PD.ClientID from PermanentDiscounts as PD
            inner join Discounts as D on D.DiscountID = PD.DiscountID
            where D.Valid = 1))
union
select ClientID, DiscountID, 'OneTime Discount' as TypeOfDiscount from
OneTimeDiscounts
where ClientID in (select ClientID from OneTimeDiscounts as OTD
                                        inner join Discounts as D on D.DiscountID =
OTD.DiscountID
                                        where D.Valid = 1 and OTD.ClientID in (select
PD.ClientID from PermanentDiscounts as PD
            inner join Discounts as D on D.DiscountID = PD.DiscountID
            where D.Valid = 1))
GO
```

Procedury

1. **sp_AddCategory** - procedura dodawania kategorii jeśli nie istnieje z opisem do tabeli Categories albo znalezienie kategorii o danej nazwie

```
CREATE PROCEDURE  [dbo].[sp_AddCategory](
    @categoryName varchar(50),
        @description varchar(250) = '(No description)',
        @categoryID int output
)
AS
BEGIN
    SET @categoryID = (SELECT CategoryID FROM Categories WHERE
@categoryName = CategoryName)
    IF(@categoryID IS NULL)
    BEGIN
        INSERT INTO Categories(CategoryName, Description)
        VALUES (@categoryName, @description)
                    SET @categoryID = SCOPE_IDENTITY()
    END
END
GO
```

2. **sp_AddCity** - procedura dodawania miasta jeśli nie istnieje lub znalezienie miasta o danej nazwie

```
CREATE PROCEDURE  [dbo].[sp_AddCity](
```

```
        @countryName varchar(50),
        @cityName varchar(50),
        @PostalCode varchar(6),
        @cityID int output
)
AS
BEGIN
        DECLARE @countryID int;
        BEGIN
                SET @cityID = (SELECT CityID FROM Cities WHERE CityName =
@cityName and PostalCode = @PostalCode and @countryName in (select CountryName
from Countries))
                IF(@cityID IS NULL)
                BEGIN
                        exec sp_AddCountry @countryname, @countryID output
                        INSERT INTO Cities(CountryID, CityName, PostalCode)
                        VALUES (@countryID, @cityName, @PostalCode)
                        SET @cityID = SCOPE_IDENTITY()
                END
        END

END
GO
```

3. **sp_AddClient** - dodaję klienta do tabeli Clients i zwraca dodane ClientID

```
CREATE PROCEDURE [dbo].[sp_AddClient]
        @phone varchar(50),
        @email varchar(50),
        @clientID int output
AS
BEGIN

   DECLARE
      @ErrorMessage NVARCHAR(4000),
      @ErrorSeverity INT,
      @ErrorState INT;

   BEGIN TRY

                SET @clientID = (select ClientID from Clients where Phone = @phone)
                IF (@clientID is not null)
                begin
                        raiserror('Phone already used',1,1)
```

```
                end

                SET @clientID = (select ClientID from Clients where Email = @email)
                IF (@clientID is not null)
                begin
                        raiserror('Email already used',1,1)
                end

                insert into Clients (Phone, Email)
                values (@phone, @email)
                set @clientID = SCOPE_IDENTITY()

  END TRY
  BEGIN CATCH
    SELECT
      @ErrorMessage = ERROR_MESSAGE(),
      @ErrorSeverity = ERROR_SEVERITY(),
      @ErrorState = ERROR_STATE();
    RAISERROR(@ErrorMessage, @ErrorSeverity, @ErrorState);
  END CATCH

END
GO
```

4. **sp_AddCompany** - dodaję firmę do bazy danych

```
CREATE PROCEDURE [dbo].[sp_AddCompany]
        @companyName varchar(50),
        @nip varchar(50),
        @phone varchar(50),
        @email varchar(50),
        @address varchar(max),
        @postalCode varchar(50),
        @cityName varchar(50),
        @countryName varchar(50)
AS
BEGIN
  DECLARE
    @ErrorMessage NVARCHAR(4000),
    @ErrorSeverity INT,
    @ErrorState INT,
                @clientID int;
```

```
   BEGIN TRY
                SET @clientID = (select ClientID from Companies where NIP = @nip)
                IF (@clientID is not null)
                begin
                        raiserror('Company with given NIP is already in database',1,1)
                end


      DECLARE @cityID int
                exec sp_AddCity @countryName, @cityName, @postalCode, @cityID
output
                exec sp_AddClient @phone, @email, @clientID output
                insert into Companies (ClientID, CompanyName, NIP, Address, CityID)
                values (@clientID, @companyName, @nip, @address, @cityID)

   END TRY
   BEGIN CATCH
      SELECT
         @ErrorMessage = ERROR_MESSAGE(),
         @ErrorSeverity = ERROR_SEVERITY(),
         @ErrorState = ERROR_STATE();
      RAISERROR(@ErrorMessage, @ErrorSeverity, @ErrorState);
   END CATCH
END

GO
```

5. **sp_AddCompanyReservationEmployeeName** - procedura dodania imienia do szczegółów rezerwacji firmy

```
CREATE PROCEDURE [dbo].[sp_AddCompanyReservationEmployeeName]
        @Name varchar(50),
        @DetailID int
AS
BEGIN
        insert into CompanyReservationEmployeeNames (Name, DetailID)
        values (@Name, @DetailID)
END
GO
```

6. **sp_AddCountry** - procedura dodania państwa lub znalezienie państwa o danej nazwie

```
CREATE PROCEDURE  [dbo].[sp_AddCountry](
	@countryName varchar(50),
	@countryID int output
)
AS
BEGIN
	SET @countryID = (SELECT CountryID FROM Countries WHERE CountryName =
@countryName)
	IF(@countryID IS NULL)
	BEGIN

	INSERT INTO Countries(CountryName)
	VALUES (@countryName)
	SET @countryID = SCOPE_IDENTITY()
	END
END
GO
```

7. **sp_AddDiscount** - procedura dodania nowej zniżki lub znalezienie zniżki o podanej dacie rozpoczęcia naliczania zniżki

```
CREATE PROCEDURE [dbo].[sp_AddDiscount]
	@collectingStartDate datetime,
	@valid int = 0,
	@discountID int output
AS
BEGIN
	INSERT INTO Discounts (CollectingStartDate, Valid)
	VALUES (@collectingStartDate, @valid)
	SET @discountID = SCOPE_IDENTITY()
END
GO
```

8. **sp_AddEmployee** - procedura dodania nowego pracowanika

```
CREATE PROCEDURE [dbo].[sp_AddEmployee]
	@firstName varchar(50),
	@lastName varchar(50),
	@phone varchar(50),
	@countryName varchar(50),
	@cityName varchar(50),
```

```
        @PostalCode varchar(6),
        @bossID int

AS
BEGIN
        -- SET NOCOUNT ON added to prevent extra result sets from
        -- interfering with SELECT statements.
        SET NOCOUNT ON;

        DECLARE
                @ErrorMessage NVARCHAR(4000),
                @ErrorSeverity INT,
                @ErrorState INT;

   BEGIN TRY
        DECLARE @cityID int
                exec sp_AddCity @countryName, @cityName, @PostalCode, @cityID
output
                insert into Employees (FirstName, LastName, Phone, BossID, CityID)
                VALUES (@firstName, @lastName, @phone, @bossID, @cityID)
        END TRY
        BEGIN CATCH
                SELECT
                        @ErrorMessage = ERROR_MESSAGE(),
                        @ErrorSeverity = ERROR_SEVERITY(),
                        @ErrorState = ERROR_STATE();
                RAISERROR(@ErrorMessage, @ErrorSeverity, @ErrorState);
        END CATCH
END
GO
```

9. **sp_AddIndividualClient** - procedura dodania nowego klienta indywiduanego

```
CREATE PROCEDURE [dbo].[sp_AddIndividualClient]
        @firstName varchar(50),
        @lastName varchar(50),
        @phone varchar(50),
        @email varchar(50)
AS
BEGIN
        DECLARE
      @ErrorMessage NVARCHAR(4000),
      @ErrorSeverity INT,
```

```
    @ErrorState INT;

  BEGIN TRY
            DECLARE
                @permanenetDiscountID int,
                @oneTimeDiscountID int,
                @clientID int

            exec sp_AddClient @phone, @email, @clientID output
            insert into IndividualClients (ClientID, FirstName, LastName)
            values (@clientID, @firstName, @lastName)

            exec sp_AddPermanentDiscount @clientID, @permanenetDiscountID
output
            exec sp_AddOneTimeDiscount @clientID, @oneTimeDiscountID output

  END TRY
  BEGIN CATCH
    SELECT
      @ErrorMessage = ERROR_MESSAGE(),
      @ErrorSeverity = ERROR_SEVERITY(),
      @ErrorState = ERROR_STATE();
    RAISERROR(@ErrorMessage, @ErrorSeverity, @ErrorState);
  END CATCH
END
GO
```

10. sp_AddInvoice - procedura dodania nowej faktury

```
CREATE PROCEDURE [dbo].[sp_AddInvoice]
      @companyID int,
      @invoiceID int output,
      @issuedDate datetime2 = null
AS
BEGIN
      begin try
            begin transaction

            if (@issuedDate is null)
                  set @issuedDate = getdate()


            insert into Invoices (CompanyID, IssuedDate)
```

```
                values (@companyID, @issuedDate)


                set @invoiceID = SCOPE_IDENTITY()


                commit
        end try
        begin catch
                if @@TRANCOUNT > 0 rollback
        end catch
END
GO
```

11. **sp_AddMeal** - procedura dodania nowego posiłku lub znalezienie posiłku o danej nazwie i kategorii

```
CREATE PROCEDURE  [dbo].[sp_AddMeal](
      @mealName varchar(50),
            @categoryName varchar(50),
            @mealID int output
)
AS
BEGIN
        DECLARE @categoryID int

   SET @mealID = (SELECT MealID FROM Meals WHERE @mealName = MealName)

   IF(@mealID IS NULL)
        BEGIN
                exec sp_AddCategory @categoryName, '(No description)', @categoryID
output
                print @categoryID
                INSERT INTO Meals(CategoryID, MealName)
                VALUES (@categoryID, @mealName)
                SET @mealID = SCOPE_IDENTITY()
        END
END
GO
```

12. **sp_AddMenuPosition** - dodanie nowej pozycji w menu lub znalezienie pozycji w menu o danej nazwie posiłku, dacie rozpoczęcia pozycji i ceny

```
CREATE PROCEDURE [dbo].[sp_AddMenuPosition](
   --ZMIENNE
                @MealName varchar(50),
          @StartDate datetime,
          @EndDate datetime = null,
          @Price money,
          @RecordID int output
)
AS
BEGIN
   -- SET NOCOUNT ON added to prevent extra result sets from
   -- interfering with SELECT statements.
   SET NOCOUNT ON;

   DECLARE
      @ErrorMessage NVARCHAR(4000),
      @ErrorSeverity INT,
      @ErrorState INT;

   BEGIN TRY
         BEGIN TRANSACTION




      --PROCEDURA
                DECLARE @categoryName varchar(50), @MealID int;
                exec sp_AddMeal @MealName, @categoryName, @MealID output


                IF(@StartDate<DATEADD(day, 1, GETDATE()))
                 BEGIN
                        RAISERROR('Menu position must be added at least 1 day in
advance.', 16, 1);
                 END

                IF ((SELECT RecordID FROM Menu WHERE MealID = @MealID and
(EndDate is NULL or EndDate > @StartDate) and (@EndDate is NULL or @EndDate >
StartDate)) is not null)
                BEGIN
                        RAISERROR('This meal is already on the menu during this time.',
16, 1);
                END
```

```
                INSERT INTO Menu(MealID, StartDate, EndDate, Price )
                VALUES (@MealID, @StartDate, @EndDate, @Price)
                SET @RecordID = SCOPE_IDENTITY()



        COMMIT TRANSACTION
        END TRY
   BEGIN CATCH
                IF @@TRANCOUNT > 0 ROLLBACK;
      SELECT
        @ErrorMessage = ERROR_MESSAGE(),
        @ErrorSeverity = ERROR_SEVERITY(),
        @ErrorState = ERROR_STATE();
      RAISERROR(@ErrorMessage, @ErrorSeverity, @ErrorState);
   END CATCH
END
GO
```

13. sp_AddOneTimeDiscount- dodaję jednorazową zniżkę do klienta

```
CREATE PROCEDURE [dbo].[sp_AddOneTimeDiscount]
        @clientID int,
        @discountID int output,
        @startDate datetime2 = null
AS
BEGIN
        DECLARE @now datetime
        if (@startDate is not null)
                set @now = @startDate
        else
                set @now = GETDATE()
        exec sp_AddDiscount @now, 0, @discountID output
        INSERT INTO OneTimeDiscounts (DiscountID, ClientID, StartDate, ExpirationDate)
        VALUES (@discountID, @clientID, null, null)

END
GO
```

14. sp_AddOneTimeDiscountsConstants - Dodaję nową stałą dla zniżek, która zaczyna obowiązywać od danej daty, aż do wstawienia następnej wartości

```
CREATE PROCEDURE [dbo].[sp_AddOneTimeDiscountsConstants](
   --ZMIENNE
   @ParameterName varchar(50),
   @StartDate datetime,
   @Value decimal(18,2)
)
AS
BEGIN
   -- SET NOCOUNT ON added to prevent extra result sets from
   -- interfering with SELECT statements.
   SET NOCOUNT ON;

   DECLARE
      @ErrorMessage NVARCHAR(4000),
      @ErrorSeverity INT,
      @ErrorState INT;

   BEGIN TRY

      --PROCEDURA
      DECLARE @ParametrID int;
      SET @ParametrID = (SELECT ParameterID FROM OneTimeDiscountParameters
WHERE ParameterName = @ParameterName)

                  -- Zaznacza zakończenie ważności poprzedniego parametru
      DECLARE @ConstantID int;
      SET @ConstantID = (SELECT ConstantsID FROM OneTimeDiscountsConstants
WHERE ParameterID = @ParametrID and EndDate is null)

      UPDATE OneTimeDiscountsConstants SET EndDate = @StartDate where
ConstantsID = @ConstantID

      INSERT INTO OneTimeDiscountsConstants(ParameterID, StartDate, EndDate, Value)
      VALUES (@ParametrID, @StartDate, null, @Value)



   END TRY
   BEGIN CATCH
      SELECT
         @ErrorMessage = ERROR_MESSAGE(),
         @ErrorSeverity = ERROR_SEVERITY(),
         @ErrorState = ERROR_STATE();
      RAISERROR(@ErrorMessage, @ErrorSeverity, @ErrorState);
   END CATCH
```

```
END
GO
```

15. **sp_AddOrder** - Dodaję zamówienie na listę dań dla podanego klienta, definiując czy również czy zamówienie ma być na wynos

```
CREATE PROCEDURE [dbo].[sp_AddOrder](
	@orderedDate as datetime2 = null,
	@requiredDate as datetime2,
	@clientID as int = null,
	@takeOut as bit,
	@payingLater as bit,
	@mealNameList as MealNameList READONLY,
	@orderID int output
	-- find/add order
)
AS
BEGIN
   SET NOCOUNT ON;

   DECLARE
      @ErrorMessage NVARCHAR(4000),
      @ErrorSeverity INT,
      @ErrorState INT;

         BEGIN TRANSACTION;
   BEGIN TRY
                IF(@orderedDate IS NULL)
                BEGIN
                        SET @orderedDate = GETDATE();
                END

                print(@orderedDate)
                set @orderID = (select OrderID from Orders where @orderedDate =
OrderedDate and @clientID = ClientID)


                if (@orderID is null)
                begin
                        -- search for employee with the smallest number of Orders
                        declare @employeeID int;
```

```
                        set @employeeID = (select top 1 E.EmployeeID from Employees as
E inner join Orders as O on E.EmployeeID = O.EmployeeID group by E.EmployeeID order
by count(*));

                        -- search for one time discount --(if null)-> the permanent discount
--(if null) -> discountid = null
                        if(@clientID is not null)
                        begin
                                declare @discountID int;
                                set @discountID = (select DiscountID from
OneTimeDiscounts where ClientID = @clientID);

                                if(@discountID is null)
                                begin
                                        set @discountID = (select DiscountID from
PermanentDiscounts where ClientID = @clientID);
                                end
                        end

                        -- check if client has paid now
                        declare @paidDate datetime2;
                        if(@payingLater = 0)
                        begin
                                set @paidDate = DATEADD(second, 1, GETDATE());
                        end

                        print(@paidDate)


                        INSERT INTO Orders(EmployeeID, OrderedDate, RequiredDate,
ServedDate, PaidDate, ClientID, TakeOut, PayingLater, DiscountID)
        VALUES (@employeeID, @orderedDate, @requiredDate, null, @paidDate,
@clientID, @takeOut, @payingLater, @discountID)


                        SET @orderID = SCOPE_IDENTITY();

                        EXEC sp_AddOrderDetails @orderID, @orderedDate,
@requiredDate,@mealNameList

                end
        COMMIT;
   END TRY
   BEGIN CATCH
                IF @@TRANCOUNT > 0 ROLLBACK;
```

```
    SELECT
      @ErrorMessage = ERROR_MESSAGE(),
      @ErrorSeverity = ERROR_SEVERITY(),
      @ErrorState = ERROR_STATE();
    RAISERROR(@ErrorMessage, @ErrorSeverity, @ErrorState);
  END CATCH
END
GO
```

16. **sp_AddOrderDetails** - dodaję listę zamówionych dań i ich ilości dla danego zamówienia

```
CREATE PROCEDURE [dbo].[sp_AddOrderDetails](
      @orderID as int,
      @orderedDate as datetime2,
      @requiredDate as datetime2,
  @mealNameList as MealNameList READONLY
)
AS
BEGIN
  SET NOCOUNT ON;

  DECLARE
    @ErrorMessage NVARCHAR(4000),
    @ErrorSeverity INT,
    @ErrorState INT;

      declare CursorMealNames CURSOR FOR (select * from @mealNameList);
      declare @mealName as varchar(max);
      declare @quantity as int;

  BEGIN TRY
      BEGIN TRANSACTION
            -- search meals and inesrt things into the table
            OPEN CursorMealNames;
            FETCH NEXT FROM CursorMealNames INTO @mealName, @quantity;

            DECLARE @RecordID as int;
            DECLARE @IsSeafood as bit = 0;

            WHILE @@FETCH_STATUS = 0
            BEGIN
                  --PRINT @mealName;
```

```
                        --PRINT @quantity;
                        SET @RecordID = null;
                        SET @RecordID = (select top 1 RecordID from
MenuWithRecordIDView where @mealName = MealName)

                        IF(@RecordID is not null) -- it is in current menu
                        BEGIN
                                --print(@RecordID)
                                -- check is meal seafood
                                SET @IsSeafood = (select top 1 IIF(CategoryName =
'seafood', 1, 0) from Categories where CategoryID = (select CategoryID from Meals where
MealName = @mealName))
                                IF(@IsSeafood = 1)
                                BEGIN
                                        -- check Required date -> should be on Th, Fr, Sat
                                        IF(5 <= DATEPART(dw, @requiredDate) and
DATEPART(dw, @requiredDate) <= 7)
                                        BEGIN
                                                -- check OrderedDate -> should be before
Monday's week of requireddate
                                                IF((select top 1 IIF(DATEPART(wk,
@requiredDate) > DATEPART(wk, @orderedDate), 1, 0)) = 0)
                                                BEGIN
                                                        RAISERROR('OrderedDate should be
before Monday week of RequiredDate for seafood.',
                                                                16,
                                                                1
                                                        );
                                                END
                                        END
                                        ELSE
                                        BEGIN
                                        RAISERROR('RequiredDate should be on Thursday,
Friday or Saturday for seafood.',
                                                                16,
                                                                1
                                                        );
                                        END
                                END

                                INSERT INTO OrderDetails(OrderID, RecordID, Quantity)
                                VALUES (@orderID, @RecordID, @quantity)

                        END
```

```
                        FETCH NEXT FROM CursorMealNames into @mealName,
@quantity;
                END

                CLOSE CursorMealNames;
                DEALLOCATE CursorMealNames;

        COMMIT TRANSACTION
   END TRY
   BEGIN CATCH
                ROLLBACK TRANSACTION
      SELECT
        @ErrorMessage = ERROR_MESSAGE(),
        @ErrorSeverity = ERROR_SEVERITY(),
        @ErrorState = ERROR_STATE();
      RAISERROR(@ErrorMessage, @ErrorSeverity, @ErrorState);
   END CATCH
END
GO
```

17. sp_AddPermanentDiscount - dodaję dożywotnią zniżkę dla klienta

```
CREATE PROCEDURE [dbo].[sp_AddPermanentDiscount]
        @clientID int,
        @discountID int output
AS
BEGIN
        SET @discountID = (select DiscountID from PermanentDiscounts where ClientID =
@clientID)
        IF (@discountID is null)
        begin
                DECLARE @now datetime
                set @now = GETDATE()
                exec sp_AddDiscount @now, 0, @discountID output
                INSERT INTO PermanentDiscounts (DiscountID,ClientID)
                VALUES (@discountID,@clientID)
                SET @discountID = SCOPE_IDENTITY()
        end
END
GO
```

18. **sp_AddPernamentDiscountsConstants** - dodanie stałych dla dożywotnich zniżek, które zaczynają działać od danego momentu, aż do następnej zmiany

```
CREATE PROCEDURE [dbo].[sp_AddPernamentDiscountsConstants](
   --ZMIENNE
         @ParameterName varchar(50),
           @StartDate datetime,
           @Value decimal(18,2)
)
AS
BEGIN
   -- SET NOCOUNT ON added to prevent extra result sets from
   -- interfering with SELECT statements.
   SET NOCOUNT ON;

   DECLARE
      @ErrorMessage NVARCHAR(4000),
      @ErrorSeverity INT,
      @ErrorState INT;

   BEGIN TRY




      --PROCEDURA
                  DECLARE @ParametrID int;
                  SET @ParametrID = (SELECT ParameterID FROM
PermamentDiscountsParameters WHERE ParameterName = @ParameterName)

                  -- Zaznacza zakończenie ważności poprzedniego parametru
                  DECLARE @ConstantID int;
                  SET @ConstantID = (SELECT ConstantsID FROM
PermanentDiscountsConstants WHERE ParameterID = @ParametrID and EndDate is null)

                  UPDATE PermanentDiscountsConstants SET EndDate = @StartDate where
ConstantsID = @ConstantID


                  INSERT INTO PermanentDiscountsConstants(ParameterID, StartDate,
EndDate, Value)
                  VALUES (@ParametrID, @StartDate, null, @Value)
```

```
        END TRY
    BEGIN CATCH
        SELECT
            @ErrorMessage = ERROR_MESSAGE(),
            @ErrorSeverity = ERROR_SEVERITY(),
            @ErrorState = ERROR_STATE();
        RAISERROR(@ErrorMessage, @ErrorSeverity, @ErrorState);
    END CATCH
END
GO
```

19. sp_AddReservation - dodaję rezerwację i zwraca dodane ReservationID

```
CREATE PROCEDURE [dbo].[sp_AddReservation]
        @startDate datetime2,
        @reservationID int output,
        @endDate datetime = null
AS
BEGIN
        DECLARE
    @ErrorMessage NVARCHAR(4000),
    @ErrorSeverity INT,
    @ErrorState INT;


    BEGIN TRY
                BEGIN TRANSACTION;

                declare @statusCode int
                set @statusCode = (select StatusCode from ReservationStatuses where
Description = 'Waiting')

                insert into Reservations (StartDate, EndDate, StatusCode)
                values (@startDate, @endDate, @statusCode)

                set @reservationID = SCOPE_IDENTITY()

                COMMIT;
    END TRY
    BEGIN CATCH
```

```
                IF @@TRANCOUNT > 0 ROLLBACK;
        SELECT
            @ErrorMessage = ERROR_MESSAGE(),
            @ErrorSeverity = ERROR_SEVERITY(),
            @ErrorState = ERROR_STATE();
        RAISERROR(@ErrorMessage, @ErrorSeverity, @ErrorState);
    END CATCH
END
GO
```

20. sp_AddTable - procedura dodania nowego stołu

```
CREATE PROCEDURE [dbo].[sp_AddTable]
        @seatsAmount INT
AS
BEGIN
        -- SET NOCOUNT ON added to prevent extra result sets from
        -- interfering with SELECT statements.
        SET NOCOUNT ON;

    -- Insert statements for procedure here
        BEGIN TRY
                INSERT INTO Tables (SeatsAmount)
                VALUES (@seatsAmount)
        END TRY
        BEGIN CATCH
                DECLARE @error varchar(max)
                SET @error = 'Invalid value of seatsAmount: ' + cast(@seatsAmount as
varchar)
                RAISERROR(@error,1,1)
        END CATCH
END
GO
```

21. sp_AddTableToIndividualReservation - dodanie stolika do rezerwacji indywidualnego klienta

```
CREATE PROCEDURE [dbo].[sp_AddTableToIndividualReservation](
        @reservationID as int,
        @tableID as int
)
AS
```

```
BEGIN
   SET NOCOUNT ON;

        DECLARE @amountOfPeople INT = (select AmountOfPeople from
IndividualReservations where ReservationID = @reservationID);

   DECLARE
      @ErrorMessage NVARCHAR(4000),
      @ErrorSeverity INT,
      @ErrorState INT;

        BEGIN TRANSACTION;
   BEGIN TRY
                DECLARE @seatsAmount INT = (select SeatsAmount from Tables where
TableID = @tableID)

                IF(@seatsAmount < @amountOfPeople)
                BEGIN
                        RAISERROR('Number of seats is too small.',
                                                                16,
                                                                1
                                                );
                END

                DECLARE @startDate datetime, @endDate datetime;
                SET @startDate = (SELECT StartDate from Reservations where
ReservationID = @reservationID)
                SET @endDate = (SELECT EndDate from Reservations where
ReservationID = @reservationID)

                IF (@EndDate is null)
                BEGIN
                        SET @EndDate = DATEADD(hour, 12, @StartDate)
                        UPDATE Reservations SET EndDate = @EndDate where
ReservationID = @ReservationID
                END

                IF (@tableID not in (select TableID from GetEmptyTables(@startDate,
@endDate)))
                BEGIN
                        RAISERROR('Table is not available during this time.', 16, 1);
                END

                UPDATE IndividualReservations SET TableID = @tableID WHERE
ReservationID = @reservationID
```

```
		DECLARE @StatusCode int;
		SET @StatusCode = (SELECT StatusCode from ReservationStatuses
where Description = 'Accepted' )
		UPDATE Reservations SET StatusCode = @StatusCode where
ReservationID = @ReservationID

	COMMIT;
   END TRY
   BEGIN CATCH
		IF @@TRANCOUNT > 0 ROLLBACK;
      SELECT
	@ErrorMessage = ERROR_MESSAGE(),
	@ErrorSeverity = ERROR_SEVERITY(),
	@ErrorState = ERROR_STATE();
      RAISERROR(@ErrorMessage, @ErrorSeverity, @ErrorState);
   END CATCH
END
GO
```

22. **sp_IssueInvoiceMonthly** - procedura wystawienia miesięcznej faktury na dany miesiąc

```
CREATE PROCEDURE [dbo].[sp_IssueInvoiceMonthly]
	@companyID int,
	@year int,
	@month int
AS
BEGIN
	begin try
		begin transaction

		declare @invoiceID int
		exec sp_AddInvoice @companyID, @invoiceID output

		declare @OrderID int
		declare OrdersCursor cursor for(select OrderID from Orders where
year(OrderedDate)=@year and month(OrderedDate)=@month)
		open OrderCursor
		fetch next from OrderCursor into @OrderID
```

```
		while @@FETCH_STATUS = 0
		begin

			insert into InvoiceDetails (InvoiceID, OrderID)
			values (@invoiceID, @OrderID)

			fetch next from OrderCursor into @OrderID

		end

		close OrderCursor
		deallocate OrderCursor

		commit
	end try
	begin catch
		if @@TRANCOUNT > 0 rollback
	end catch
END
GO
```

23. **sp_AddTableToIndividualReservation** - dodanie stolika do rezerwacji indywidualnego klienta

```
create PROCEDURE [dbo].[sp_AddTableToIndividualReservation](
	@reservationID as int,
	@tableID as int
)
AS
BEGIN
   SET NOCOUNT ON;

	DECLARE @amountOfPeople INT = (select AmountOfPeople from
IndividualReservations where ReservationID = @reservationID);

   DECLARE
      @ErrorMessage NVARCHAR(4000),
      @ErrorSeverity INT,
      @ErrorState INT;

	BEGIN TRANSACTION;
   BEGIN TRY
```

```
                DECLARE @seatsAmount INT = (select SeatsAmount from Tables where
TableID = @tableID)

                IF(@seatsAmount < @amountOfPeople)
                BEGIN
                        RAISERROR('Number of seats is too small.',
                                                                16,
                                                                1
                                                );
                END
                ELSE
                BEGIN
                        UPDATE IndividualReservations SET TableID = @tableID WHERE
ReservationID = @reservationID
                END
        COMMIT;
   END TRY
   BEGIN CATCH
                IF @@TRANCOUNT > 0 ROLLBACK;
      SELECT
         @ErrorMessage = ERROR_MESSAGE(),
         @ErrorSeverity = ERROR_SEVERITY(),
         @ErrorState = ERROR_STATE();
      RAISERROR(@ErrorMessage, @ErrorSeverity, @ErrorState);
   END CATCH
END
GO
```

24. sp_DeleteMenuPosition - procedura usuwająca pozycję z menu

```
CREATE PROCEDURE [dbo].[sp_DeleteMenuPosition](
   --ZMIENNE
          @RecordID int
)
AS
BEGIN
   -- SET NOCOUNT ON added to prevent extra result sets from
   -- interfering with SELECT statements.
   SET NOCOUNT ON;

   DECLARE
      @ErrorMessage NVARCHAR(4000),
      @ErrorSeverity INT,
```

```
    @ErrorState INT;

  BEGIN TRY
      BEGIN TRANSACTION




    --PROCEDURA
            DECLARE @StartDate datetime, @EndDate datetime;

            SET @StartDate = (SELECT StartDate from Menu where RecordID =
@RecordID)



            IF (@StartDate > DATEADD(day, 1, GETDATE()))
            BEGIN
                  DELETE Menu where RecordID = @RecordID
            END
            ELSE
            BEGIN
                  SET @EndDate = (SELECT EndDate from Menu where RecordID =
@RecordID)
                  IF (ISNULL(@EndDate, DATEADD(day, 1, GETDATE())) >=
DATEADD(day, 1, GETDATE()) )
                  BEGIN
                        UPDATE Menu SET EndDate = DATEADD(day, 1,
GETDATE()) where RecordID = @RecordID
                  END
                  ELSE
                  BEGIN
                        RAISERROR('Menu position cannot be deleted.', 16, 1);
                  END
            END




      COMMIT TRANSACTION
      END TRY
  BEGIN CATCH
            IF @@TRANCOUNT > 0 ROLLBACK;
    SELECT
      @ErrorMessage = ERROR_MESSAGE(),
      @ErrorSeverity = ERROR_SEVERITY(),
```

```
        @ErrorState = ERROR_STATE();
    RAISERROR(@ErrorMessage, @ErrorSeverity, @ErrorState);
  END CATCH
END
GO
```

25. sp_CancelReservation- procedura odrzucająca rezerwację (dla managera)

```
CREATE PROCEDURE [dbo].[sp_CancelReservation](
  --ZMIENNE
        @ReservationID int
)
AS
BEGIN
  -- SET NOCOUNT ON added to prevent extra result sets from
  -- interfering with SELECT statements.
  SET NOCOUNT ON;

  DECLARE
    @ErrorMessage NVARCHAR(4000),
    @ErrorSeverity INT,
    @ErrorState INT;

  BEGIN TRY
      BEGIN TRANSACTION

            -- Ustawiam Status na odrzucony
            DECLARE @StatusCode int;
            SET @StatusCode = (SELECT StatusCode from ReservationStatuses
where Description = 'Canceled' )
            UPDATE Reservations SET StatusCode = @StatusCode where
ReservationID = @ReservationID


      COMMIT TRANSACTION
      END TRY
  BEGIN CATCH
            IF @@TRANCOUNT > 0 ROLLBACK;
    SELECT
      @ErrorMessage = ERROR_MESSAGE(),
      @ErrorSeverity = ERROR_SEVERITY(),
      @ErrorState = ERROR_STATE();
    RAISERROR(@ErrorMessage, @ErrorSeverity, @ErrorState);
```

```
   END CATCH
END
GO
```

26. sp_IssueInvoiceOrder - procedura wystawienia faktury dla danej firmy

```
CREATE PROCEDURE [dbo].[sp_IssueInvoiceOrder]
        @companyID int,
        @OrderID int
AS
BEGIN
        begin try
                begin transaction

                        declare @invoiceID int
                        exec sp_AddInvoice @companyID, @invoiceID output

                        insert into InvoiceDetails (InvoiceID, OrderID)
                        values (@invoiceID, @OrderID)

                commit
        end try
        begin catch
                if @@TRANCOUNT > 0 rollback
        end catch

END
GO
```

27. sp_MakeCompanyReservation - procedura, dzięki której firma może złożyć rezerwację

```
CREATE PROCEDURE [dbo].[sp_MakeCompanyReservation](
   --ZMIENNE
          @CompanyName varchar(50),
          @StartDate datetime,
          @EndDate datetime = null,
          @AmountOfPeople int,
          @Names as NamesList readonly,
          @DivisionAmount as DivisionList readonly,
          @ReservationID int output
```

```
)
AS
BEGIN
   -- SET NOCOUNT ON added to prevent extra result sets from
   -- interfering with SELECT statements.
   SET NOCOUNT ON;

   DECLARE
      @ErrorMessage NVARCHAR(4000),
      @ErrorSeverity INT,
      @ErrorState INT;

   BEGIN TRY
         BEGIN TRANSACTION

                  -- Sprawdza ilość ludzi
                  IF(@amountOfPeople < 2)
                  BEGIN
                           RAISERROR('Number of people is too small.',
                                                                        16,
                                                                        1
                                                      );
                  END

                  --Dodanie rezerwacji do głównej tabeli
                  exec sp_AddReservation @startDate, @reservationID output, @endDate

                  -- Dodaje do CompanyReservation (na razie wszystkie ordery na null)
                  DECLARE @ClientID int;
                  SET @ClientID = (SELECT ClientID FROM Companies where
CompanyName = @CompanyName)

                  INSERT INTO CompanyReservations(ReservationID, ClientID, OrderID)
                           VALUES (@ReservationID, @ClientID, NULL)


                  -- Dodaje do CompanyReservationDetails, zawsze bez stolika
                  declare @peopleNamesAmount int, @peopleDivisionAmount int


                  set @peopleNamesAmount = (select count(*) from @Names)
                  set @peopleDivisionAmount = (select sum(Amount) from @DivisionAmount)
                  if (@peopleNamesAmount != @peopleDivisionAmount)
                  begin
                           raiserror('Amount of names do not match division amounts', 1, 1)
```

```
        end

        declare DivisionListCursor cursor for (select * from @DivisionAmount)
        declare NamesListCursor cursor for (select * from @Names)
        declare @amount int
        open NamesListCursor
        open DivisionListCursor

        declare @counter int
        declare @name varchar(50)
        declare @detailID int

        fetch next from DivisionListCursor into @amount
        while @@FETCH_STATUS = 0
        begin

                insert into CompanyReservationDetails (ReservationID,
AmountOfPeople)
                values (@ReservationID, @amount)

                set @detailID = SCOPE_IDENTITY()


                set @counter = 0
                while @counter < @amount
                begin
                        fetch next from NamesListCursor into @name

                        insert into CompanyReservationEmployeeNames (DetailID,
Name)
                        values (@detailID, @name)

                        set @counter = @counter +1
                end

                fetch next from DivisionListCursor into @amount

        end

        close NamesListCursor
        deallocate NamesListCursor
        close DivisionListCursor
        deallocate DivisionListCursor
```

```
        COMMIT TRANSACTION
        END TRY
    BEGIN CATCH
                IF @@TRANCOUNT > 0 ROLLBACK;
        SELECT
        @ErrorMessage = ERROR_MESSAGE(),
        @ErrorSeverity = ERROR_SEVERITY(),
        @ErrorState = ERROR_STATE();
        RAISERROR(@ErrorMessage, @ErrorSeverity, @ErrorState);
    END CATCH
END
GO
```

28. **sp_MakeIndividualReservation** - procedura, dzięki której indywidualny klient może złożyć rezerwację

```
CREATE PROCEDURE [dbo].[sp_MakeIndividualReservation](
        @orderedDate as datetime2 = null,
        @requiredDate as datetime2,
        @clientID as int,
        @payingLater as bit,
        @mealNameList as MealNameList READONLY,
        @amountOfPeople as int,
        @reservationID as int output,
        @startDate datetime2 = null,
        @endDate datetime2 = null
)
AS
BEGIN
    SET NOCOUNT ON;

        DECLARE @WZ INT = (select RP.Value from ReservationParameters as RP where
ParameterName = 'WZ');
        DECLARE @WK INT = (select RP.Value from ReservationParameters as RP where
ParameterName = 'WK');
        DECLARE @orderID INT;

    DECLARE
        @ErrorMessage NVARCHAR(4000),
        @ErrorSeverity INT,
        @ErrorState INT;
```

```
        BEGIN TRANSACTION;
    BEGIN TRY
                -- check amountofpeople
                IF(@amountOfPeople < 2)
                BEGIN
                        RAISERROR('Number of people is too small.',
                                                                16,
                                                                1
                                                        );
                END


                -- check WK
                IF(ISNULL((select NumberOfOrders from IndividualNumberOfOrdersView
where ClientID = @clientID),0) < @WK)
                BEGIN
                        RAISERROR('Number of orders is too small.',
                                                                16,
                                                                1
                                                        );
                END

                -- add order
                exec sp_AddOrder @orderedDate, @requiredDate, @clientID, 0,
@payingLater, @mealNameList, @orderID output

                -- check if reservation for that order already exists
                IF((select 1 from IndividualReservations where OrderID = @orderID) = 1)
                BEGIN
                RAISERROR('The order already exists.',
                                                                16,
                                                                1
                                                        );
                END

                -- check WZ
                IF(@orderID is not null)
                BEGIN
                        IF(ISNULL((select TotalCost from OrdersTotalCostsView where
OrderID = @orderID), 0)< @WZ)
                        BEGIN
                        RAISERROR('Cost of order is too small.',
                                                                16,
                                                                1
                                                        );
```

```
                        END

                        if (@startDate is null)
                        set @startDate = @requiredDate

                        exec sp_AddReservation @startDate, @reservationID output,
@endDate
                        INSERT INTO IndividualReservations(ReservationID, OrderID,
ClientID, TableID, AmountOfPeople)
        VALUES (@reservationID, @orderID, @clientID, null, @amountOfPeople)

                END
        COMMIT;
    END TRY
    BEGIN CATCH
                IF @@TRANCOUNT > 0 ROLLBACK;
        SELECT
            @ErrorMessage = ERROR_MESSAGE(),
            @ErrorSeverity = ERROR_SEVERITY(),
            @ErrorState = ERROR_STATE();
        RAISERROR(@ErrorMessage, @ErrorSeverity, @ErrorState);
    END CATCH
END
GO
```

29. **sp_SetTablesForCompanyReservation** - procedura, która przyznaje stolik dla oczekującej rezerwacji, oraz akceptuje ją (dla managera)

```
CREATE PROCEDURE [dbo].[sp_SetTablesForCompanyReservation](
    --ZMIENNE
            @ReservationID int,
            @ChosenTables ChosenTablesForCompanyList READONLY
)
AS
BEGIN
    -- SET NOCOUNT ON added to prevent extra result sets from
    -- interfering with SELECT statements.
    SET NOCOUNT ON;

    DECLARE
        @ErrorMessage NVARCHAR(4000),
        @ErrorSeverity INT,
        @ErrorState INT;
```

```
BEGIN TRY
    BEGIN TRANSACTION



  --PROCEDURA

        DECLARE @StartDate datetime, @EndDate datetime;
        SET @StartDate = (SELECT StartDate from Reservations where
ReservationID = @ReservationID)
        SET @EndDate = (SELECT EndDate from Reservations where
ReservationID = @ReservationID)

        IF (@EndDate is null)
        BEGIN
            SET @EndDate = DATEADD(hour, 12, @StartDate)
            UPDATE Reservations SET EndDate = @EndDate where
ReservationID = @ReservationID
        END

        -- Dodaje stoliki z listy
        DECLARE CursorChosenTables CURSOR FOR (select * from
@ChosenTables);
        DECLARE @TableID int, @AmountOfPeople int;

        OPEN CursorChosenTables;
        FETCH NEXT FROM CursorChosenTables INTO @TableID,
@AmountOfPeople;
        WHILE (@@FETCH_STATUS = 0)
        BEGIN

            IF (@TableID not in (select TableID from
GetEmptyTables(@StartDate, @EndDate)))
            BEGIN
                RAISERROR('Table is not available during this time.', 16, 1);
            END

            -- Dodaje do CompanyReservationDetails
            INSERT INTO CompanyReservationDetails(ReservationID, TableID,
AmountOfPeople)
            VALUES (@ReservationID, @TableID, @AmountOfPeople)
```

```
                        FETCH NEXT FROM CursorChosenTables INTO @TableID,
@AmountOfPeople;
                END
                CLOSE CursorChosenTables;
                DEALLOCATE CursorChosenTables;

                -- Ustawiam Status na zaakceptowane, dodaje datę zakonczenia jeśli jej
wcześniej nie było
                DECLARE @StatusCode int;
                SET @StatusCode = (SELECT StatusCode from ReservationStatuses
where Description = 'Accepted' )
                UPDATE Reservations SET StatusCode = @StatusCode where
ReservationID = @ReservationID



        COMMIT TRANSACTION
        END TRY
   BEGIN CATCH
                IF @@TRANCOUNT > 0 ROLLBACK;
      SELECT
        @ErrorMessage = ERROR_MESSAGE(),
        @ErrorSeverity = ERROR_SEVERITY(),
        @ErrorState = ERROR_STATE();
      RAISERROR(@ErrorMessage, @ErrorSeverity, @ErrorState);
   END CATCH
END
GO
```

30. **sp_UpdateReservationParameters** - procedura zmieniająca Parametry Rezerwacji (WZ, WK)

```
CREATE PROCEDURE [dbo].[sp_UpdateReservationParameters](
   --ZMIENNE
        @ParameterName varchar(50),
          @Value decimal(18,2)
)
AS
BEGIN
   -- SET NOCOUNT ON added to prevent extra result sets from
   -- interfering with SELECT statements.
   SET NOCOUNT ON;
```

```
   DECLARE
      @ErrorMessage NVARCHAR(4000),
      @ErrorSeverity INT,
      @ErrorState INT;

   BEGIN TRY

                  UPDATE ReservationParameters SET Value = @Value where
ParameterName = @ParameterName

         END TRY
   BEGIN CATCH
      SELECT
         @ErrorMessage = ERROR_MESSAGE(),
         @ErrorSeverity = ERROR_SEVERITY(),
         @ErrorState = ERROR_STATE();
      RAISERROR(@ErrorMessage, @ErrorSeverity, @ErrorState);
   END CATCH
END
GO
```

31. sp_AddOrderToCompanyReservation - dodaje zamówienie do rezerwacji firmowej

```
CREATE PROCEDURE [dbo].[sp_AddOrderToCompanyReservation](
         @reservationID int,
         @payingLater as bit,
         @mealNameList as MealNameList READONLY,
         @orderedDate as datetime,
         @requiredDate as datetime,
         @clientID as int,
         @orderID as int output
)
AS
BEGIN
   SET NOCOUNT ON;


   DECLARE
      @ErrorMessage NVARCHAR(4000),
      @ErrorSeverity INT,
      @ErrorState INT;

         BEGIN TRANSACTION;
   BEGIN TRY
```

```
		-- add order

		exec sp_AddOrder @orderedDate, @requiredDate, @clientID, 0,
@payingLater, @mealNameList, @orderID output

		-- update reservation
		UPDATE CompanyReservations SET OrderID = @orderID where
ReservationID = @reservationID


	COMMIT TRANSACTION;
   END TRY
   BEGIN CATCH
		IF @@TRANCOUNT > 0 ROLLBACK;
      SELECT
	@ErrorMessage = ERROR_MESSAGE(),
	@ErrorSeverity = ERROR_SEVERITY(),
	@ErrorState = ERROR_STATE();
      RAISERROR(@ErrorMessage, @ErrorSeverity, @ErrorState);
   END CATCH
END
GO
```

Funkcje

1. ***GetEmptyTables*** - funkcja zwracająca wszystkie wolne stoliki wraz z ilością miejsc, w danym terminie

```
CREATE FUNCTION [dbo].[GetEmptyTables]
(
   @StartDate datetime,
   @EndDate datetime = null
)
RETURNS TABLE
AS
RETURN
(
   select * from AllTablesView where TableID not in(Select Distinct TableID from
AllReservedTablesView
	where (StartDate <= @StartDate and EndDate >= @StartDate) or (StartDate <=
@EndDate and StartDate >= @StartDate))

)
```

```
GO
```

2. ***GetOneTimeDiscountParameterValue*** - funkcja zwracająca wartość parametru w danym czasie, dla zniżki jednorazowej

```
CREATE FUNCTION [dbo].[GetOneTimeDiscountParameterValue]
(
   @ParametrName varchar(50),
   @Date datetime
)
RETURNS TABLE
AS
RETURN
(
   select Value from OneTimeDiscountsConstants
         where ParameterID = (SELECT ParameterID from OneTimeDiscountParameters
where ParameterName = @ParametrName)
         and @Date between StartDate and isnull(EndDate, DATEADD(hour, 1,
GETDATE()))
)
```

3. ***GetPermanentDiscountParameterValue*** - funkcja zwracająca wartość parametru w danym czasie, dla zniżki jednorazowej

```
CREATE FUNCTION [dbo].[GetPermanentDiscountParameterValue]
(
   @ParametrName varchar(50),
   @Date datetime
)
RETURNS TABLE
AS
RETURN
(
   select Value from PermanentDiscountsConstants
         where ParameterID = (SELECT ParameterID from
PermamentDiscountsParameters where ParameterName = @ParametrName)
         and @Date between StartDate and isnull(EndDate, DATEADD(hour, 1,
GETDATE()))
)
```

4. ***AmountOfMenuPositionsToChange***- funkcja zwracająca ilość potrzebnych do zrobienia zmian w menu

```
CREATE FUNCTION [dbo].[AmountOfMenuPositionsToChange]
```

```
(
)
RETURNS int
AS
BEGIN
	-- Declare the return variable here
	DECLARE @amount int

	-- Add the T-SQL statements to compute the return value here
	declare @fromDate datetime
	set @fromDate = DATEADD(WEEK, -2, GETDATE())

	declare @startingAmount int,
			@endedAmount int,
			@startedAmount int

	set @startingAmount = (select count(*) from Menu where StartDate < @fromDate
and ISNULL(EndDate, getdate()) >= @fromDate)
	set @endedAmount = (select count(*) from Menu where EndDate is not null and
@fromDate < EndDate)
	set @startedAmount = (select count(*) from Menu where @fromDate < StartDate)

	-- Return the result of the function
	declare @min int
	if (@endedAmount < @startedAmount)
	begin
		set @min = @endedAmount
	end
	else
	begin
		set @min = @startedAmount
	end


	set @amount = @startedAmount - @min
	return @amount

END
GO
```

Indexy

1. **orderID_index**

```
create index orderID_index on Orders(OrderID)

create index orderID_index on OrderDetails(OrderID)
```

2. **parameterID_index**

```
create index parameterID_index on PermamentDiscountsParameters(ParameterID)

create index parameterID_index on OneTimeDiscountParameters(ParameterID)

create index parameterID_index on ReservationParameters(ParameterID)
```

3. **countryID_index**

```
create index countryID_index on Countries(CountryID)
```

4. **cityID_index**

```
create index cityID_index on Cities(CityID)
```

5. **countryName_index**

```
create index countryName_index on Countries(CountryName)
```

6. **employeeID_index**

```
create index employeeID_index on Employees(EmployeeID)
```

7. **invoiceID_index**

```
create index invoiceID_index on Invoices(InvoiceID)
```

8. **constantsID_index**

```
create index constantsID_index on PermanentDiscountsConstants(ConstantsID)

create index constantsID_index on OneTimeDiscountsConstants(ConstantsID)
```

9. **discountID_index**

```
create index discountID_index on PermanentDiscounts(DiscountID)
```

```
create index discountID_index on Discounts(DiscountID)

create index discountID_index on OneTimeDiscounts(DiscountID)
```

10. clientID_index

```
create index clientID_index on Clients(ClientID)

create index clientID_index on IndividualClients(ClientID)

create index clientID_index on Companies(ClientID)
```

11. reservationID_index

```
create index reservationID_index on Reservations(ReservationID)

create index reservationID_index on IndividualReservations(ReservationID)

create index reservationID_index on CompanyReservations(ReservationID)
```

12. statusCode_index

```
create index statusCode_index on ReservationStatuses(StatusCode)
```

13. tableID_index

```
create index tableID_index on Tables(TableID)
```

14. nameID_index

```
create index nameID_index on CompanyReservationEmployeeNames(NameID)
```

15. detailID_index

```
create index detailID_index on CompanyReservationDetails(DetailID)
```

16. recordID_index

```
create index recordID_index on Menu(RecordID)
```

17. mealID_index

```
create index mealID_index on Meals(MealID)
```

18. categoryID_index

```
create index categoryID_index on Categories(CategoryID)
```

19. categoryName_index

```
create index categoryName_index on Categories(CategoryName)
```

20. mealName_index

```
create index mealName_index on  Meals(MealName)
```

Triggery

1. **tr_UpdateClientDiscounts**

```
CREATE TRIGGER [dbo].[tr_UpdateClientDiscounts]
   ON  [dbo].[Orders]
   AFTER INSERT
AS
BEGIN
          -- SET NOCOUNT ON added to prevent extra result sets from
          -- interfering with SELECT statements.
          SET NOCOUNT ON;

    -- Insert statements for trigger here

          DECLARE
       @ErrorMessage NVARCHAR(4000),
       @ErrorSeverity INT,
       @ErrorState INT;

          BEGIN TRY
    BEGIN TRANSACTION

                    declare @clientID int
```

```
	set @clientID = (select ClientID from inserted)

	if (@clientID is not null)
	begin
		--check if permament discount is already valid
		declare @permamentValid int, @permamentCollectingStartDate datetime2
		set @permamentValid = (select valid from PermanentDiscounts inner join
Discounts on Discounts.DiscountID = PermanentDiscounts.DiscountID where
ClientID=@clientID)
		set @permamentCollectingStartDate = (select CollectingStartDate from
PermanentDiscounts inner join Discounts on Discounts.DiscountID =
PermanentDiscounts.DiscountID where ClientID=@clientID)
		if (@permamentValid = 0)
		begin
			--update permament discount if neccessary
			declare @Z1 int, @K1 int
			set @Z1 = (select Value from
GetPermanentDiscountParameterValue('Z1', GETDATE()))
			set @K1 = (select Value from
GetPermanentDiscountParameterValue('K1', GETDATE()))

			if (@Z1 is null)
				raiserror('No value for Z1 is set',1,1)

			if (@K1 is null)
				raiserror('No value for K1 is set',1,1)


			declare @amountOfOrders int
			set @amountOfOrders = (select count(*) from OrdersTotalCostsView
inner join Orders

		on Orders.OrderID=OrdersTotalCostsView.OrderID

		where TotalCost >= @K1 and Orders.OrderedDate >
@permamentCollectingStartDate)

			if (@amountOfOrders >= @Z1)
				update Discounts
				set Valid = 1 where DiscountID in (select DiscountID from
PermanentDiscounts where PermanentDiscounts.ClientID = @clientID)

		end

		--check and update one time discount if neccessary
```

```
		declare @oneTimeValid int, @oneTimeCollectingStartDate datetime2
		set @oneTimeValid = (select valid from OneTimeDiscounts inner join
Discounts on Discounts.DiscountID = OneTimeDiscounts.DiscountID where
ClientID=@clientID and Valid = 0)
		set @oneTimeCollectingStartDate = (select CollectingStartDate from
PermanentDiscounts inner join Discounts on Discounts.DiscountID =
PermanentDiscounts.DiscountID where ClientID=@clientID and Valid=0)
		if (@oneTimeValid = 0)
		begin
			--update permament discount if neccessary
			declare @D1 int, @K2 int
			set @D1 = (select Value from
GetOneTimeDiscountParameterValue('D1', GETDATE()))
			set @K2 = (select Value from
GetOneTimeDiscountParameterValue('K2', GETDATE()))

			if (@D1 is null)
				raiserror('No value for D1 is set',1,1)

			if (@K2 is null)
				raiserror('No value for K2 is set',1,1)



			declare @valueofOrders int
			set @valueofOrders = (select sum(TotalCost) from
OrdersTotalCostsView inner join Orders

		on OrdersTotalCostsView.OrderID=Orders.OrderID

		where ClientID=@clientID and Orders.OrderedDate >
@oneTimeCollectingStartDate)

			if (@valueofOrders >= @K2)
			begin

				update Discounts
				set Valid=1
				where DiscountID in (select DiscountID from
OneTimeDiscounts where ClientID=@clientID and Valid=0)

				update OneTimeDiscounts
				set StartDate = GETDATE(), ExpirationDate =
DATEADD(DAY, @D1, GETDATE())
				where ClientID=@clientID and StartDate is null and
ExpirationDate is null
```

```
                                --if one time discount set valid add new one for collecting
after expiration date
                                declare @discountID int, @startDate datetime2
                                set @startDate = DATEADD(DAY, @D1, GETDATE())
                                exec sp_AddOneTimeDiscount @clientID, @discountID
output, @startDate

                            end

                        end




                end

    COMMIT TRANSACTION
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0 ROLLBACK;
        SELECT
            @ErrorMessage = ERROR_MESSAGE(),
            @ErrorSeverity = ERROR_SEVERITY(),
            @ErrorState = ERROR_STATE();
        RAISERROR(@ErrorMessage, @ErrorSeverity, @ErrorState);
    END CATCH
END
GO

ALTER TABLE [dbo].[Orders] ENABLE TRIGGER [tr_UpdateClientDiscounts]
GO
```